ROBOTS IN ASSISTED
LIVING ENVIRONMENTS

UNOBTRUSIVE, EFFICIENT, RELIABLE AND MODULAR SOLUTIONS FOR INDEPENDENT AGEING

Research Innovation Action

Project Number: 643892 Start Date of Project: 01/04/2015 Duration: 36 months

DELIVERABLE 4.3

Architecture for extending smart homes with robotic platform III

Dissemination Level	**Public**
Due Date of Deliverable	Project Month 30, September 2017
Actual Submission Date	10 May 2018
Work Package	WP4, *Physical home architecture development and integration of cost-effective, reliable and power-efficient RADIO components for elder monitoring and caring*
Task	T4.1, *Designing device interconnection and interfacing*
Lead Beneficiary	RUB
Contributing beneficiaries	NCSR-D, TWG, S&C, AVN
Type	Report
Status	Submitted
Version	Final

Project funded by the European Union's Horizon 2020 Research and Innovation Actions

Abstract

Architecture document, pertaining to RADIO device interconnection and interfacing; specifications on interfacing the different domains; efficient processing of sensitive content in the distributed RADIO environment; and the orchestration of the overall system.

History and Contributors

Ver	Date	Description	Contributors
01	15 May 2017	First draft, summarizing D4.2 as a starting point and establishing new document structure.	RUB
02	19 May 2017	New event wrapper added for composite events (multiple sensors from different WSN) (update of Section 8.4)	NCSR-D, S&C
03	6 Jun 2017	System of distributed ROS nodes (new Section 7.2)	NCSR-D
05	14 Sep 2017	Task switching and other improvement in robot behaviour (new Section 9) and using the task switching mechanism in orchestration (minor update of Section 8.2).	NCSR-D
06	20 Oct 2017	Accelerating image processing algorithms (new Section 6)	RUB
07	22 Oct 2017	Hardware acceleration (new Section 4.2) and final robot interface design (new Section 2.2)	RUB
08	22 Nov 2017	Additions to Section 4 and 5	TWG, AVN
09	30 Nov 2017	Additions to Section 6 and Conclusions	TWG, AVN
10	4 May 2018	Minor updates and, in general, synchronization with development in WP4.	NCSR-D
11	9 May 2018	Internal review.	NCSR-D
Fin	10 May 2018	Final preparation and submission	NCSR-D, TWG, RUB

Abbreviations and Acronyms

BLE – Bluetooth Low Energy

ID – Identification

IoT – Internet of Things

WIFI – Wireless Fidelity

LAN – Local Area Network

API – Application Programming Interface

HCI – Host Controller Interface

LLC – Logical Link Control

ISM – Industrial Scientific Medical

SoC – System on Chip

FPGA – Field Programmable Gate Array

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Purpose and Scope

This deliverable is the physical architecture of the RADIO Home, covering RADIO device interconnection and interfacing, specifications on interfacing the different domains, and on fast and energy efficient data processing in the distributed RADIO environment.

Within the scope of this document is:

- To design the physical architecture of the RADIO Home, and especially the wireless communications architecture between the RADIO Robot platform, the Smart Home devices, and the Main Controller that make up each RADIO Home.
- To design the architecture and the policies for managing the heterogeneous computing elements of the RADIO Home, including the central server, FPGAs, and the on-board Robot controller.

Outside the scope of this document is the architecture (either conceptual or physical) of the communication between the RADIO Home and other nodes of the RADIO ecosystem, such as cloud storage components and components meant to be used by hospital personnel or informal care-givers. This will be dealt with in Task 5.1.

1.2 Approach

This deliverable documents work done in Task 4.1, which specifies and designs the interconnection structure and interfaces to exchange data between the home automation infrastructure and the robotic platform. This task also specifies the sensors and the processing units such as FPGAs, the Robot on-board computer, or other computers on the premises which comprise the *RADIO Home*, i.e., the part of the overall RADIO system that is deployed within a single home. In addition, Task 4.1 tackles the following assignments:

- Investigating the most efficient way, in terms of power and delay overhead, to process different kinds of sensor data in the distributed RADIO environment.
- Observing privacy for the user.
- Observing technical limitations such as bandwidth and processing power.
- Striving for robustness through device redundancy.

Alternative hardware and sensor positioning configurations are also investigated as part of this task with the focus on power and performance trade-offs between fixed function accelerators and more programmable (or even pure software) solutions. The programmable solutions offer more flexibility to provide several dedicated services to the end-users through software updates or extensions. **However, fixed logic hardware solutions offer the significant advantage of privacy for two main reasons: i) the sensor data is pre-processed and immediately destroyed and ii) in case that further processing is required, this is performed on anonymized data (the outcome of the pre-processing step).**

We have extended the work done in D4.2 and D4.1 for this task as follows:

- We finalized the Robot interface design for interconnecting the NUC and the PicoZed to the ROS environment.
- No changes were conducted in the interconnection domains (Section 3).
- We implemented a RADIO image processing algorithm in hardware and optimized its performance. The resource usage of the hardware block and of the complete designs are also evaluated. The implementation of the image processing algorithms was performed using three

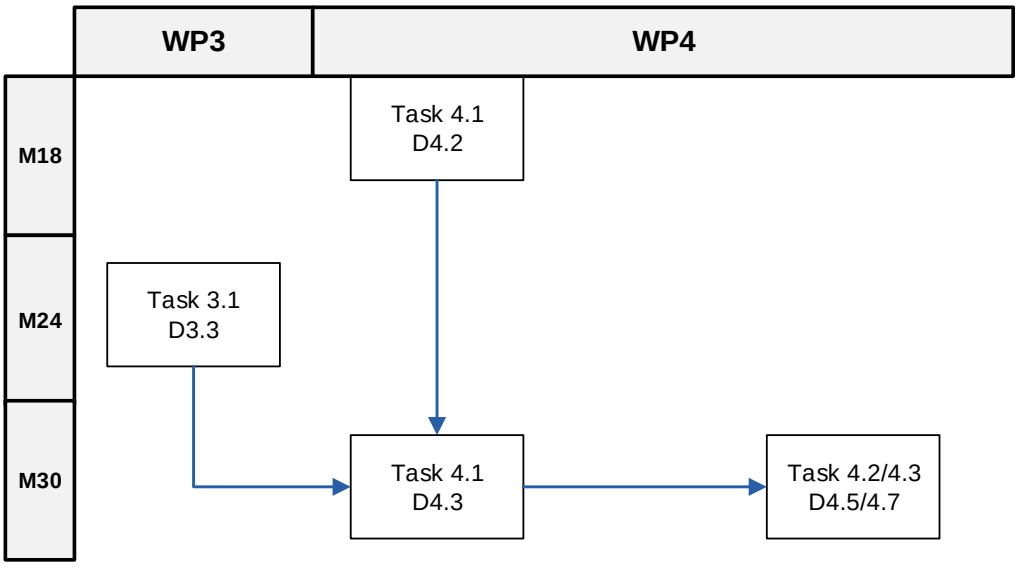

Figure 1: Relation to other Work Packages and Deliverables

different methods/tools: hardware-only implementation based on Verilog HDL code, through Xilinx Vivado HLS (high-level synthesis), and through HW-SW partitioning.

- The system level energy savings approach proposed in D4.1 and D4.2 was refined and extended to include an energy estimator tool. The energy estimation approach is verified assuming specific usage scenarios that are further described in D4.9.

The definition of the conceptual architecture in Task 4.1 has also allowed the consortium to refine the approach to WP4 as a whole and to establish the following work plan:

- **TWG** and **AVN** will design the interface for data transfer and communication among network nodes, and design hardware modules' interfaces with their respective infrastructural system components such as nodes or/and sensors.
- **TWG** and **S&C** will ensure compatibility of the RADIO-prototyped components with the rest of the system through emulation or detailed analysis.
- **RUB** and **TWG** will explore various hardware configurations and task mapping policies among all the processing units of the RADIO ecosystem in order to extract the best solution in terms of latency, power consumption, and area.
- **AVN** and **TWG** will investigate system-level power savings modes taking as input the user behavior targeting to increase the autonomy of the Robot in terms of battery charges.

1.3 Relation to other Work Packages and Deliverables

This document is the third in a series of closely related deliverables. The final version due in M30 (September 2017) is used to synchronize Task 4.1 with Task 4.2 and Task 4.3. The final version (M30) documents the architecture and interfacing of the final hardware components and robotic platform.

This deliverable updates, extends deliverable D4.2. Because this is the final document of Task 4.1, it will also summarize all progress from the deliverable D4.1 and D4.2. Task 4.2 and Task 4.3 use the physical architecture developed in this deliverable for the final prototype of the RADIO architecture.

2 DEVICE INTERCONNECTION AND INTERFACING

This chapter specifies the interconnection between the different devices within the smart home environment and defines how the respective devices interface with the smart home infrastructure.

2.1 Summary of D4.1 and D4.2

As introduced in deliverable D4.1, the smart home is comprised of several devices with different communication protocols. Figure 2 shows all available devices within the smart home environment and their respective interconnections.

As seen in Figure 2, the robot requires WIFI and Bluetooth connectivity. The gateway is a Raspberry Pi and requires Bluetooth, Z-Wave, and LAN interfaces. Because Bluetooth connectivity is required for both the gateway and the robot, deliverable D4.1 analysed the Bluetooth protocol in depth. This analysis helped drive development of the BLE communication. Several device options were also considered for usage on the gateway and the robot. These devices also served for debug purposes and easier development. Since the gateway functionality is implemented on a Raspberry Pi, the Z-Wave interface for the gateway is provided by the Razberry module for the Raspberry Pi. The Z-Wave devices are only able to communicate with the Z-Wave gateway and an external server, the IoT platform. The positions of the BLE devices and the Z-Wave devices were assumed to be in fixed positions. This potentially allowed the annotation of the robots map with the BLE devices.

In deliverable D4.2, the BLE devices of the smart home environment should also support mobile BLE devices. These devices should be locatable by the robot. Therefore, accurate measurements of the received signal strength indication (RSSI) were required. When using the integrated Bluetooth and WIFI chip of the NUC, inconsistent results were achieved for the RSSI values at fixed positions. The cause of this was that the WIFI and Bluetooth signal interfered each other when the NUCs chip was used for both protocols simultaneously, see Figure 3.

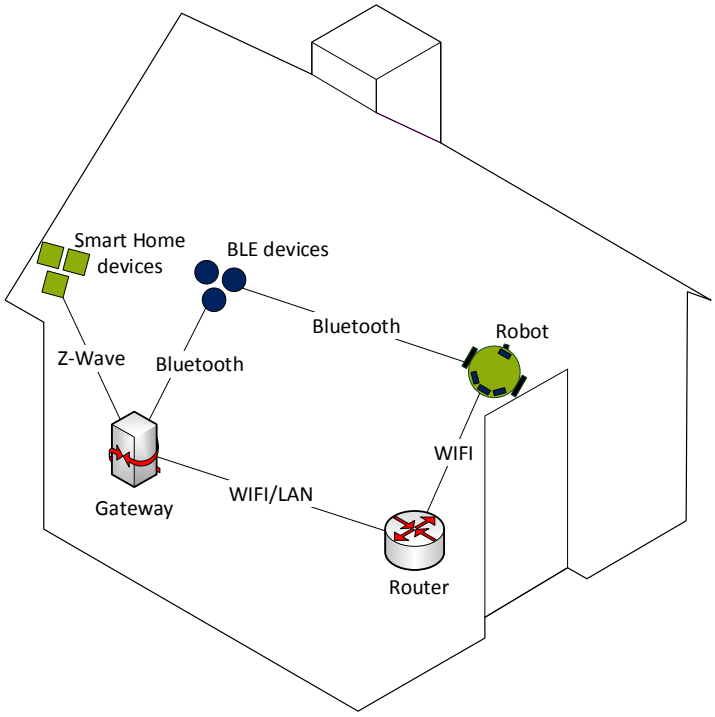

Figure 2: Device interconnection within the smart home environment

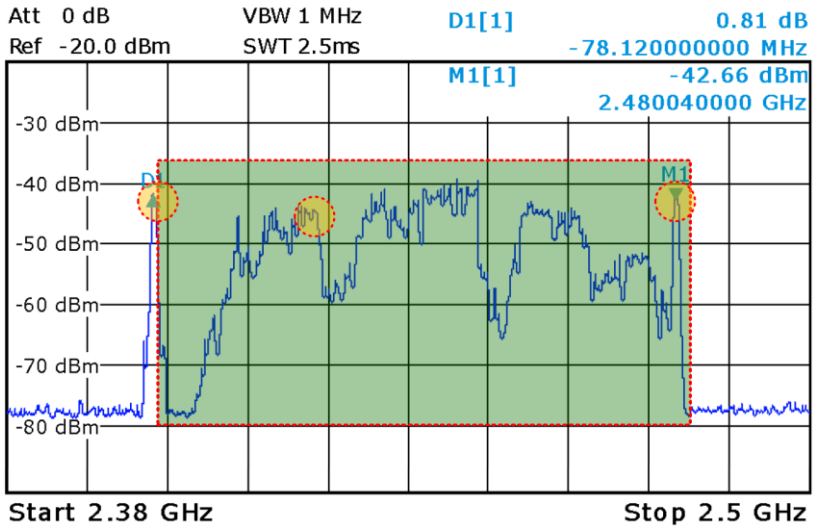

Figure 3 Spectrum of BLE and WIFI with interference

In deliverable D4.2, the Z-Wave devices are now able to communicate with the other smart home devices and manage the network. Management of the network includes adding and removing devices, controlling the networks routing. These are handled by "function" commands. The Z-Wave device functions are controlled through "command" commands. Generally, the "command" command are either for user or for device configuration. Table 1 shows the commands that are accessible by external applications.

Table 1: Z-Wave commands and their type

External accessible functions	
Z-Wave command	Type of Class
sendData	FUNCTION_CLASS
AddNodetoNetwork	FUNCTION_CLASS
RemoveNodeFromNetwork	FUNCTION_CLASS
setValue	FUNCTION_CLASS
SetNodeLocation	FUNCTION_CLASS
SetNodeName	FUNCTION_CLASS
toggleActuatorSensor	COMMAND_CLASS
toggleDimmableSensor	COMMAND_CLASS
setThermostatSetPoint	COMMAND_CLASS
setThermostatMode	COMMAND_CLASS
setThermostatFanMode	COMMAND_CLASS

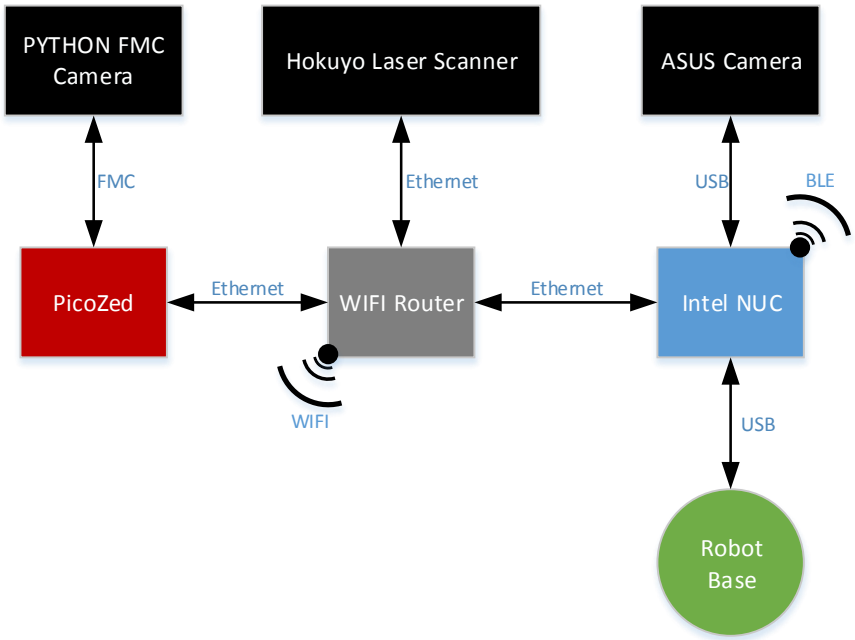

Figure 4 Final device interconnection of the robot

2.2 Final Robot Interface Design

The main interface components of the robot is a wireless router form ASUS (see Figure 4 Final device interconnection of the robot). The router has connections to the Intel NUC, the PicoZed, and the Hokuyo Laser Scanner via Ethernet cables. This enables fast communication between all connected components. Additionally, no additional work is required for bridging an internet connection from the PicoZed to the NUC, thus increasing connection stability. Additionally, the employment of the router removed the strong interference between the BLE and WIFI signals, because both interfaces are not used on the same chip and they are now spatially separated. Furthermore, the usage of a router enables the usage of the 5 GHz WIFI band in case further signal interferences with BLE and Z-Wave occur.

3 SPECIFICATIONS ON INTERFACING THE DIFFERENT DOMAINS

This chapter deals with the challenge of transferring data through several different protocol domains. No changes have occurred in this chapter for deliverable D4.3.

3.1 Summary of D4.1 and D4.2

In deliverable D4.1, we analyzed the interfacing requirements between the different protocol domains. Based on the results from Chapter 2, the protocol domains Bluetooth, Z-Wave, and WIFI were analyzed.

Table 2 shows the summary of the specification for the cross domain interfaces for the Bluetooth domain.

Table 2: Specification for the cross domain interfaces for the Bluetooth domain

Bluetooth Domain Specification		
Cross domain interfaces	**Bluetooth – Z-Wave**	**Bluetooth – WIFI/LAN**
Necessity	not required	required
Participating entities	Smart Home gateways	Robot platform Smart Home gateways
Information exchange	commands for device manipulation	None position, type of device, functionality, payload

The Bluetooth domain interfaces with two different entities, the smart home gateway and the robot platform. While the Bluetooth-Z-Wave cross domain interface is not required, the Bluetooth-WIFI cross domain interface is required, in order to make the generated information by the Bluetooth devices available for the caregivers or the end-users. This can be either the position of the Bluetooth device, or context sensitive information.

Table 3 shows the summary of the specification for the cross domain interfaces for the Z-Wave domain.

Table 3: Specification for the cross domain interfaces for the Z-Wave domain

Z-Wave Domain Specification		
Cross domain interfaces	**Z-Wave – Bluetooth**	**Z-Wave – WIFI/LAN**
Necessity	not required	required
Participating entities	Smart Home gateways	Smart Home gateways
Information exchange	commands for device manipulation	position, type of device, functionality, payload

The Z-Wave devices only interface with the smart home gateway. Therefore, it is possible for the Z-Wave devices to communicate with the Bluetooth devices. In the context of the RADIO project, this is not required. Because the Z-Wave devices need to communicate and receive commands from the IoT platform, the Z-Wave-WIFI cross domain interface is required in order to ensure full functionality of the Z-Wave devices.

Table 4 shows the summary of the specification for the cross domain interface of the WIFI/LAN domain.

Table 4: Specification for the cross domain interfaces for the WIFI/LAN domain

WIFI/LAN Domain Specification		
Cross domain interfaces	**WIFI/LAN – Z-Wave**	**WIFI/LAN – Bluetooth**
Necessity	Required	required
Participating entities	Smart Home gateways	Robot platform Smart Home gateways
Information exchange	position, type of device, functionality, payload	None position, type of device, functionality, payload

Although the WIFI/LAN domain is only present between the gateways, the router and the robot platform, it is the most important domain, because it enables to relay the gathered information by each entity to the caregivers or the IoT platform. Therefore, the WIFI/LAN-Z-Wave cross domain interface and the WIFI/LAN-Bluetooth cross domain interface is required for the RADIO smart home environment.

In deliverable D4.2, no changes had to be made in terms of the specification of the cross domain interfaces.

4 FAST AND ENERGY EFFICIENT DATA PROCESSING

This chapter identifies the RADIO algorithms, which benefit from hardware acceleration. The corresponding hardware architecture is also introduced. Low power operation is also considered in the hardware architecture.

4.1 Summary of D4.1 and D4.2

In general, there are two types of data processed in the system:

- High throughput streaming data created from continually receiving the output of a microphone (audio stream) or a camera (video stream)
- Event or control-like data of relatively small size, collected by sensors. Event/measurement data can also be the outcome of streaming data analysis, e.g., processing of video can lead to the generation of an "exit" event if the camera looks towards the door

The event/measurement data can be transferred within the smart home since their payload is small. The communication protocols used in the RADIO ecosystem do not have the bandwidth to transfer raw data streams such as image or audio streams continuously between different entities for processing. Additionally, transferring image or audio streams and not processing them locally poses a security risk. Therefore, the processing of the data streams should be performed locally on the robots PicoZed FPGA, see Figure 5. The PicoZed FPGA consists of an ARM Dual Core, a Neon vector processing engine and programmable hardware. The goal is to efficiently utilize all the available resources.

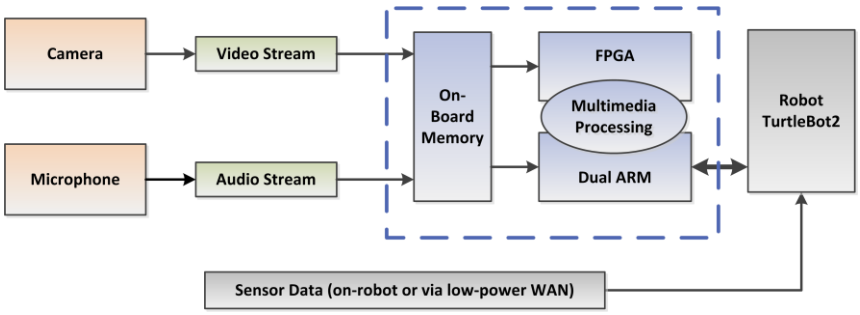

Figure 5 A high level view of the on robot processing nodes

The camera and microphone are directly connected to the FPGA platform, which then performs the pre- and post-processing tasks. The camera data streams will be continuously monitored by the processing elements of the FPGA platform and when activity is detected the corresponding algorithms (which can analyse and recognise the activity) will be triggered. Depending on the specific combination of algorithms that get triggered, some or all computational tasks may be executed in the processor (ARM cores) or accelerated with fixed logic or reconfigurable hardware components inserted in the FPGA reprogrammable logic. The algorithms required for both types of data streams can then be divided into a hardware and software component with the help of hardware software co-design tools.

Possible hardware software co-design tools were identified as valgrind[1], oprofile[2], and vampir[3]. By combining these three tools, exploitation of instruction level parallelism is possible. Because several algorithms will run simultaneously on the RADIO robot platform, priorities of the different algorithms have to be considered. If the result of the algorithm is required immediately, the processing platform has to assign more processing resources to the respective algorithm. This requires scheduling algorithms that handle dynamically appearing tasks and static tasks. These challenges define the attributes of the task scheduler of the FPGA platform.

[1] Valgrind Developers. http://valgrind.org, date of access: August 2015.

[2] OpenSource project. http://oprofile.sourceforge.net, date of access: August 2015.

[3] GWT-TUD GmbH. https://www.vampir.eu, date of access: August 2015.

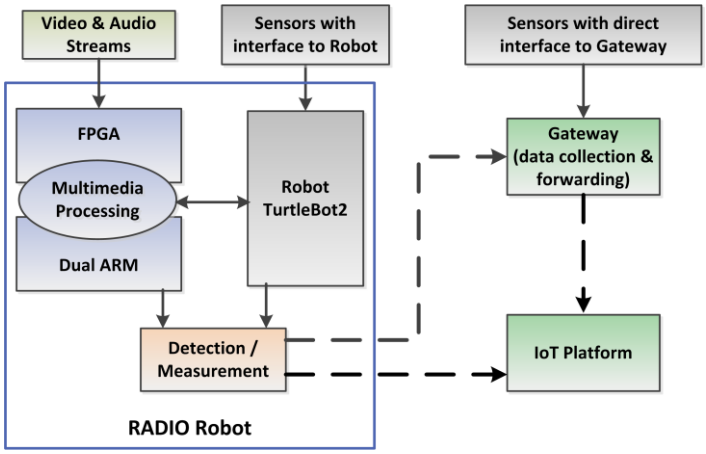

Figure 6 Diagram illustrating the data flow between the ROBOT, gateway and IoT platform

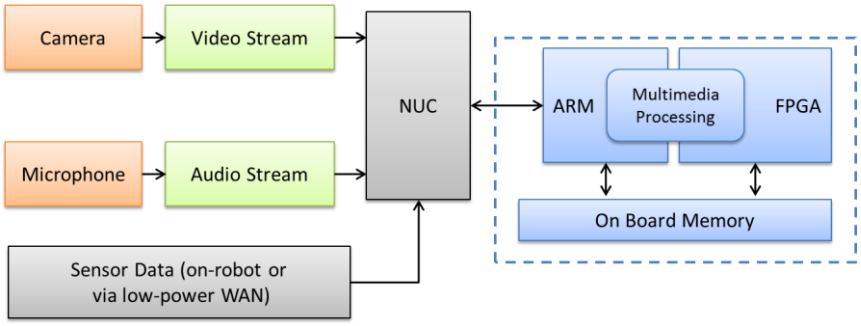

Figure 7 A high level view of the on robot processing nodes in D4.2

The last topic covered in deliverable D4.1 is the distributed RADIO environment. The RADIO home environment is connected with remote elements of the RADIO ecosystem, like the IoT platform, through its gateway. The only exception to this rule is the RADIO robot which should have the possibility of directly communicating with the IoT platform and with the RADIO gateway if necessary, see Figure 6.

The distributed sensors are usually connected directly to the gateway. A few sensors provide data important to the robot. These sensors are then connected to the robot, which then sends the data either to the gateway or directly to the IoT platform. In summary, Table relates the various data processing and transfer interface to the available domains.

In deliverable D4.2, the processing of the data stream model was updated to incorporate the Intel NUC as data aggregation platform, see Figure 7.

Table 5 Data Transfer and Process in relation to interfacing domains

Data	WIFI/LAN	Z-Wave	Bluetooth
Sensor data needed for analysis of audio and/or video streams		(optional)	X
ADL and mood recognition event log generated by the analysis of streams	X		
Sensor data that can be directly forwarded for remote processing		X	Optional
Robot location and status data	X		

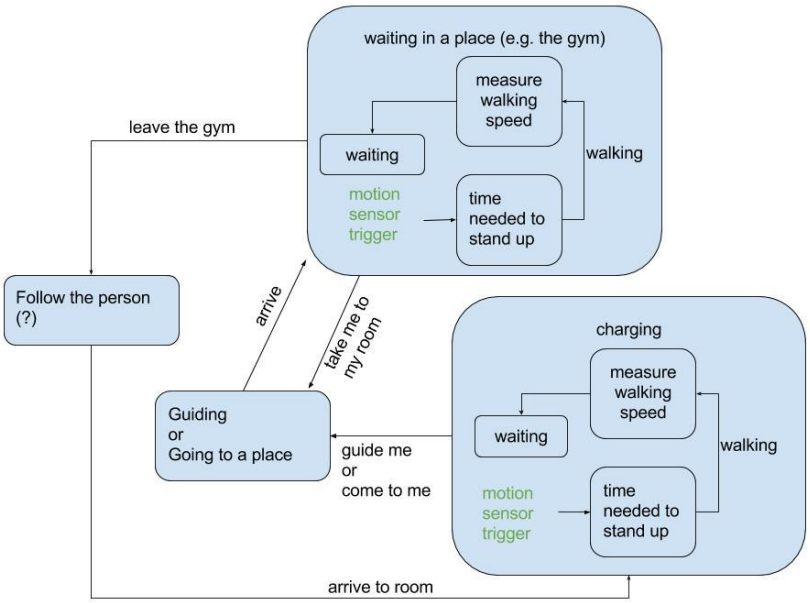

Figure 8 State diagram highlighting different (power) modes

The Intel NUC receives the data from the camera and the microphone, since the data is also required for localization and robot mapping. The Intel NUC then sends the data directly through ROS channels to the PicoZed.

Deliverable D4.2 also expanded the concept of the distributed RADIO environment. The requirements of the RADIO environment are responsiveness and efficiency. Responsiveness means that the system reacts to stimulate in a certain amount of time. The time may vary depending on the stimulus. An exemplary state diagram of the robot is shown in Figure 8.

This diagram highlights two points with green fonts where the smart home infrastructure triggers an entering event through a motion sensor. Several states also require large amounts of data processing that either can be performed with very low power consumption (i.e. *follow the person, guiding or going to a place, time needed to stand up(gym), measure walking speed (gym))* or is executed when the robot is stationed on its charging station (i.e. *time needed to stand up(in the room), measure walking speed (in the room)).*

4.2 Final Hardware Acceleration Architecture

The hardware design has undergone several changes within the RADIO project. The final hardware architecture now has the ability to accelerate any kind of image or signal processing algorithm sent from the NUC, provided that the hardware accelerator for the specific algorithm is actually available. The complete hardware design is shown in Figure 9.

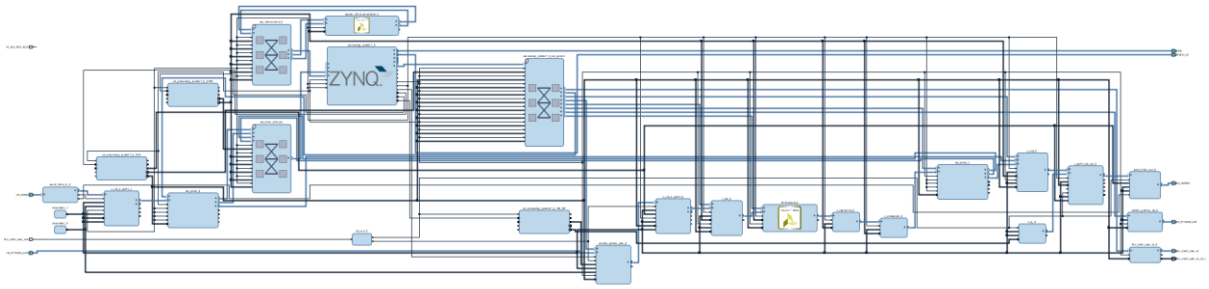

Figure 9 Complete hardware design for the RADIO FPGA

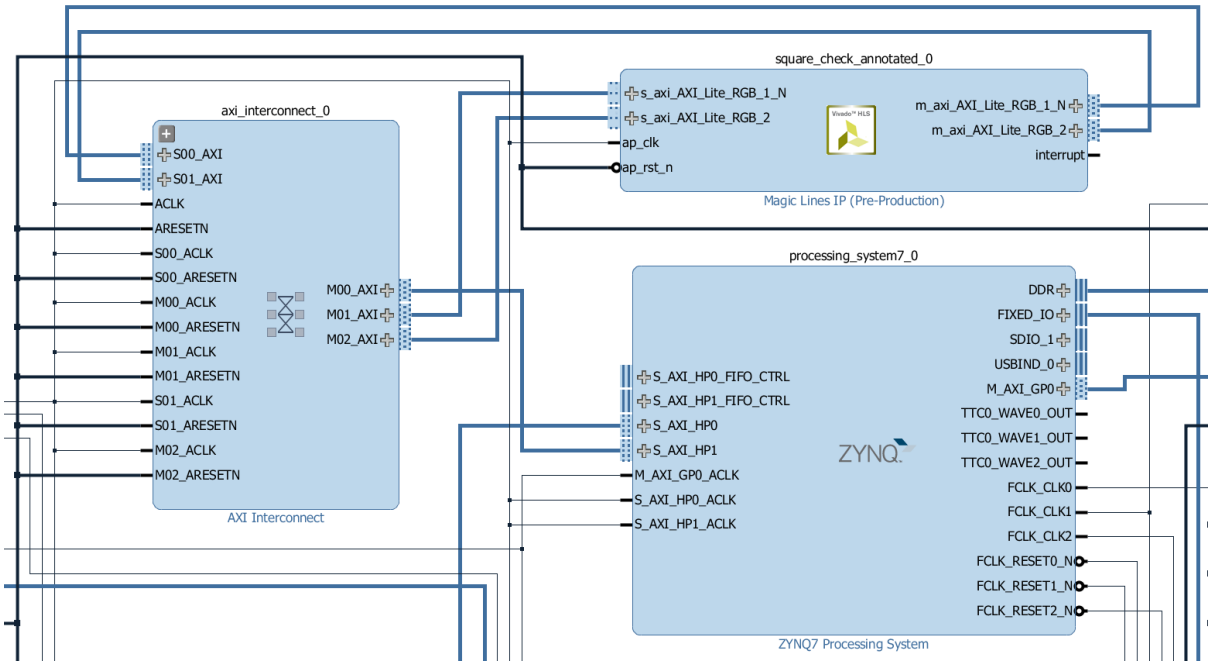

Figure 10 Hardware acceleration for image processing algorithms

This architecture supports the two function modes *hardware acceleration* and *low power operation*.

A more detailed view of the *hardware acceleration mode* design is shown in Figure 10.

The hardware accelerator is directly connected to the Zynq processing system via and AXI interconnect. The Zynq processing system receives the data from the Intel NUC over ROS messages. The payload of the ROS messages is sent to the hardware for processing. Depending on the algorithm, the hardware accelerator sends the results of the algorithm or the complete image back to the Zynq processing system for further processing. The hardware accelerator is connected to the AXI high performance port of the Zynq processing system. This allows fast data transfer between the processing system and the reconfigurable hardware which is required when performing image processing.

The *low power operation* design is depicted in Figure 11.

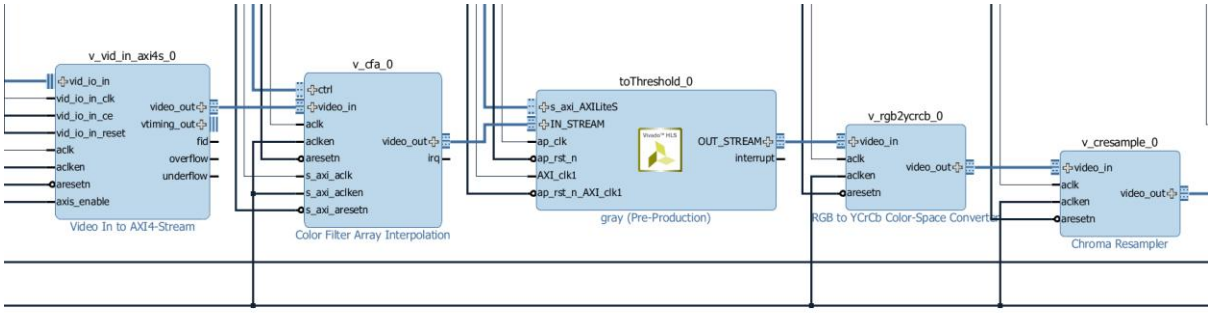

Figure 11 Image processing core in the low power operation hardware chain

In the *low power operation* mode, only the FPGA is active and performing periodically sensor update from the Python camera. This Python camera is directly connected to the FPGA hardware and therefore does not require an additional processor for transferring image data to the programmable logic. Therefore, all systems that are not required for camera usage can be put in sleep mode until the image processing core in the Python camera chain wakes up all other systems.

4.3 The Role of a dedicated HW Component

A HW accelerator component is a specially designed circuit which is implemented in FPGA (for configurability and future upgradability) and is connected directly to the other subsystems. The component is processing signals from sensors, so that simple decisions on whether other subsystems have to be employed or not can be devised. Typically, this component is equipped with the following functionality:

- Triggering mechanism, which initiates sensor data capture and processing
- Local Memory, which holds processed sensor data so that the main system RAM does not have to be used
- Signal processing acceleration functions in FPGA
- Control interfaces to turn-on and notify (or get notified by) other subsystems

The dedicated HW components are implemented in the programmable logic (PL) of the Picozed APSoC using three different techniques: hardware-only implementation based on Verilog HDL code, through Xilinx Vivado HLS (high-level synthesis), and through HW-SW partitioning. More, specifically the first type of implementation is based on the traditional way of hardware development. In particular, the implementation is written using a HDL (Verilog in this work). This approach leads to the most efficient hardware components, but it is a time-consuming approach and most importantly it cannot take full advantage of the Zynq hardware and software features e.g., the dedicated memory controllers and the dedicated busses to move data from the software part (ARM processor) to the hardware part.

On the other hand, an automated solution (e.g., our SDSoC based implementation; third hardware implementation performed in this work) is able to get advantage of the previous features and reduce significantly the development time, but the hardware modules are usually of medium quality (in terms of performance and power). As a result of this work, it was proved that the best solution is given by the Xilinx Vivado HLS (high-level synthesis) implementation. This is because, the HLS-based hardware design offers the possibility to take advantage of the various verified and fully optimized Xilinx components and as the same time it offers various parameters (in the form of pragmas in the C-code level) that can be used to optimize the design in terms of area, performance and/or power. So, based on this outcome, in the rest of this deliverable the HLS-based results (second implementation) will be mainly analyzed, since these results exhibit the best derived properties in terms of performance and power; the prime targets of the hardware component in the context of RADIO. Finally, it must be noted that all the three implementations (source files with the associated READMEs) have been uploaded in the github account of the project. More details about this can be found in D4.5.

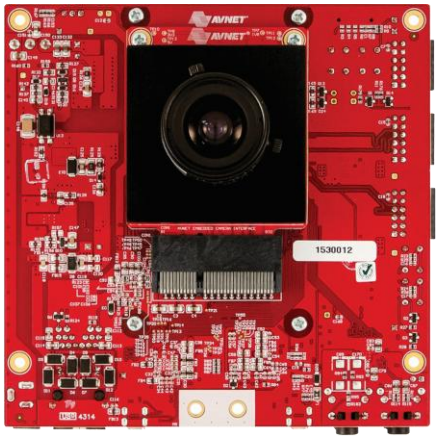

Figure 12 ON Semiconductor's PYTHON-1300 color image sensor

4.4 Visual content processing and privacy

The architecture presented Section 4.1 offers also significant advantages in terms of security and privacy. By attaching the sensors to a processing unit at the edge of the application, we are able to process the incoming data as a stream and thus there will not exist at any point in time raw images that can be read out by an attacker to compromise the data security of the application. All processing takes place in-stream or in local inaccessible memory immediately at the sensor. Only abstract derivative information is extracted from the raw data and leaves the confines of the sensor. This means that no raw visual content is stored or transmitted within the RADIO Home network when we employ edge computing devices.

The PicoZed FPGA represents such an edge computing device if an image sensor can be directly connected to the programmable logic of the PicoZed. The PicoZed consists of an ARM Dual Core, a Neon vector processing engine and the FPGA programmable hardware. A sensor, which can be directly connected to the programmable logic of the PicoZed is the PYTHON-1300 color image sensor that is depicted in Figure 12.

The PYTHON-1300 is a 1/2 inch Super-eXtended Graphics Array (SXGA) CMOS image sensor with a resolution of 1280 by 1024 pixels. It is connected to the PicoZed over the carrier card's FMC connector. This connector allows the direct connection to the programmable logic without going over the DDR memory or the ARM Dual cores of the PicoZed. Out of the box, the PYTHON-1300 is configured via software which we want to avoid in the RADIO project in order to ensure the patients visual data security. Therefore, the configuration cores of the PYTHON-1300 needed to be changed to be pre-configured in the bitstream so that the PYTHON-1300 is directly operational during the start-up of the PicoZed. To further protect the sensitive information, the pre-processing of the streaming data is performed in a fixed logic, hardware accelerator.

The PYTHON-1300 outputs images at 60 Hz. For this frequency, the image stream needs to be processed at approximately 160 MHz. Therefore, the image processing hardware accelerator for the patient's visual data security needed to be able to be clocked at the same frequency. However, the default frequency of the hardware accelerators (in general of typical FPGA designs) are generally 100 MHz or lower.

Thus, the hardware accelerator needed to be adapted to our requirements. This was done with the Vivado toolsuite from Xilinx. During hardware generation with Vivado HLS, several options exist to improve performance. Through pragma/directive usage, the time required for design space exploration is reduced.

One option to improve performance of the hardware is to replace the 32-Bit data types int and float with the 24-bit hardware efficient data types ap_int and ap_fixed. With hardware efficient data types, more operations can be executed in one cycle at the cost of additional hardware usage and less data needs to be transferred. This leads to a higher clock frequency and thus a faster overall computation time. However, by using hardware efficient data types, it is possible that the accelerator suffers from a reduced accuracy, which then would result in a larger number of iterations. With hardware efficient data types, the overall resources (DSP slices, flip flops) required by the hardware is increased. This effect is because now more logic per cycle is available through the hardware efficient data types. This leads to a faster performance but to a higher resource utilization.

Another option to improve performance is to either merge, flatten, or unroll the loops of the algorithm. The choice for flattening, unrolling or merging the loops depends heavily on the loop itself (type of loop iterators) and need to be analyzed for every loop separately. By performing either one of these three optimization steps on the respective loops, the cycles can be even further reduced.

By using Allocation Directives, it is possible to assign/map FPGA resources to specific operations. By limiting the number of available resources for the respective operations, hardware reuse can be achieved.

The end result was that through the use of these performance improvement options, we were able to set the clock frequency of the hardware accelerator to the required 160 MHz.

Another challenge we faced when implementing the visual security hardware on the PicoZed, was meeting the timing constraints of the overall design when inserting our hardware accelerator into the normal image stream flow of the PYTHON-1300. All devices need to be connected to the same clock from the same source for synchronicity reasons. However, the timing constraints of the overall design were not met when inserting our hardware accelerator and thus we were forced to clock gate several hardware blocks that did not directly influence the image stream flow over an additional clock source. This needed to be done over an additional AXI Interconnect that served as clock reference for the respective hardware blocks.

5 THE NEED FOR ENERGY EFFICIENT EXECUTION

This chapter describes a scenario, which benefits from an energy efficient hardware architecture.

5.1 Summary of D4.1 and D4.2

The RADIO robot is a unit which has many energy-hungry subsystems. These are:

- Main processor to control robot movement (NUC)
- FPGA to accelerate ADL recognition methods
- Sensors, especially the image sensor (camera)
- Mechanical subsystem (motors)
- Wireless subsystem (network)

If all subsystems are always active, the RADIO robot needs to be recharged every few hours, which results in long periods of robot non-availability. As a first step, we had to understand how each subsystem is used and if it, indeed, needs to be active at each use case. Table 6 provides an overview, assuming that robot activity can be classified in the following states:

- *Waiting:* at this state, the robot is not moving; neither is it processing sensor data. At this point, the robot is waiting to be triggered by some external event
- *Moving:* when leading the way or following a person
- *Monitoring:* at this state, the robot is not moving but it is processing sensor input data in order to detect an ADL or understand patient's mood

For some of these states, there is a difference on whether the robot is on its charging station or away of it e.g., in another room.

Table 6 Robot Subsystem Energy Usage

State	CPU	FPGA	Sensors	Motors	Network
Waiting	Used	Not used	Not used	Not used	Used
Moving	Used	Used	Used	Used	Used
Monitoring/Away	Used	Used	Used	Not used	Used
Monitoring/Charging	Used	Used	Used	Not used	Used

A more detailed description of the cases illustrated in the tables is presented below:

Waiting State: The FPGA can connect only to a ultra-low power wireless scanning device. When the user or any other RADIO system wants to instruct the robot, it should first connect to this device, and send a handshake command. This command is interpreted by the FPGA. For example, it can be used to turn-on the CPU and perform a simple action. If more complex control is needed, e.g. a user request via the tablet GUI, the CPU will turn on the network subsystem.

Table 7 Energy profile by using HW accelerators

State	CPU	FPGA	Sensors	Motors	Network
Waiting	On demand	Used	Not used	Not used	On demand
Moving	Used	Used	Used	Used	Used
Monitoring/Away	On demand	Used	Used	Not used	On demand
Monitoring/Charging	Used	Used	Used	Not used	Used

- **Monitoring/Away State**: At this state the FPGA gets triggered by external events or continuously monitors live sensor signals. Only when some (external or sensor) activity occurs, the HW

component in the FPGA will pre-process it and decide whether the CPU or/and the network subsystem has to be turned on.

- **Monitoring/Charging** and **Moving States**: At these states we may not need to employ any on-demand approach for the CPU and/or the network. However, having the dedicated hardware components in the FPGA will allow some of the processing to be offloaded there, which also yields considerable energy benefits.

The energy consumed at each state by each subsystem is not the same. For example, the CPU while waiting can be clocked at lower frequency, drastically reducing the required power. Also, sensors and FPGA can perform only basic data capture and processing when monitoring away from the charging dock and revert to full-power processing mode when this power is available.

Although a number of such techniques are used, their impact on power consumption is not drastic in all cases. To cope with this problem, our view is to develop dedicated hardware components that allow the robot to turn-off complete subsystems in some cases; turning them on only on demand and just for the short period when they are really needed. The goal is to have an improved energy profile. The results of this analysis are depicted in Table 7.

To prototype and experiment with the alternative approach discussed in this paper, we selected a small number of ADLs as target use cases for the monitoring state of the robot. The selected ADLs are the ones which detect:

- *The time needed by the patient to get out of bed:* This ADL is based on image processing algorithms that observe the patient while getting out of bed. The image processing algorithms can be parallelized availing themselves from the acceleration within the FPGA hardware. The specific algorithm divides the image into different regions. If the centre of mass of moving pixels over succeeding images lies in one of these defined regions, an event is triggered. Thus, this algorithm is able to detect if a person is sleeping, awake (but not going out of bed), and awake and standing up
- *Picking up medication cups:* The image processing methods used to detect this ADL benefit from the acceleration through the FPGA hardware as they rely on a computational intensive algorithm

As noted, the HW acceleration does not involve the complete method, but rather focuses on early detection of a high-possibility for an event so that SW-based processing can be invoked. Specifically, in the context of the above-mentioned ADLs:

- *For the time-to-stand-up ADL*, the hardware component will collect and calculate data from all regions, providing a trigger to software components when a given activity threshold is crossed
- *For the cup-detection ADL*, since this is manually triggered by the operator, hardware acceleration is not related to the recognition but to the stabilization and centering of the image. It has been observed through field trials that the robot can slightly move while waiting and this movement can issue false positives. An always running HW component will be monitoring such small movements and constantly re-centre the view

In order to determine the SW-HW co-design of the FPGA-ARM system, extensive profiling of the image processing algorithms is needed. We profiled three flow options, so that the expected benefits of various optimization approaches can be quantified, allowing focusing on these solutions that are more beneficial (in terms of power consumption) in each case. The three analyzed options are:

- *No offloading* i.e., all processing is performed on the robot's main processing unit (NUC)
- *Offload on embedded ARM core of the FPGA*; no HW acceleration
- *Offload on dedicated low-level hardware blocks in the FPGA*; ARM core can be powered down

To make a realistic profiling, we identified typical activity use cases with the help of non-technical partners of the RADIO consortium. Each typical activity is depicted as a combination of five states for the robot subsystem:

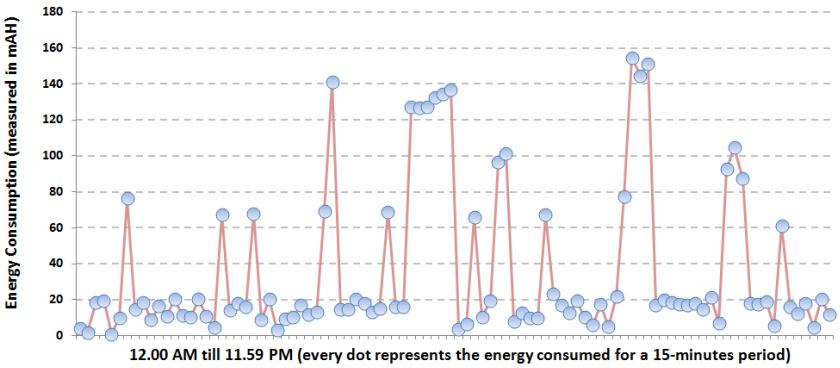

Figure 13 Daily activity profile

- *Moving*, where the robot is actually moving and uses its motor, sensors and camera
- *Monitoring*, where the robot is waiting for an event to be triggered by what it can see
- *Sensing*, where the robot is using its onboard sensors or communicates with smart home
- *Processing*, where heavy processing to analyze sensor and camera input is required
- *Idle*, where the robot is on but is doing nothing of the above

It is important to understand that a specific human activity (e.g., having lunch) will combine more than one of the above states (e.g., looking, sensing, and processing).

By accumulating the energy needs at each activity, we are able to extract the daily activity profile in terms of energy consumption as shown in Figure 13. More specifically, the data presented in Figure 13 were extracted by **i)** analyzing the daily activity patterns of the person(s) that is being monitored during the whole day (24 hours) in their domestic environment and **ii)** conducting live measurements to calculate the energy consumed in each discrete phase of the robot (consequently in each activity of the target person) assuming that all data processing in performed in NUC. Finally, we should mention that the daily activity patterns were collected by personal care-givers during the third pilot phase of the RADIO project and represent the (averaged) activity patterns of three persons.

Power-Savings Results: The three options analyzed in the previous section are then tested on the profile illustrated in Figure 13. Our target is to reveal the potential for maximizing battery life in terms of reducing the required re-charges during the day; in other words, to increase the autonomy of the AAL robot by using specialized hardware accelerators. The target areas are the points located in the lower part of Figure 13 (juxtaposed the x-axis). These points correspond to the cases in which the robot is either in the *sensing* or *idle state* waiting for an event to occur.

To this end, we performed a battery load calculation and our results are presented in Figure 14. The vertical axis in Figure 14 shows the battery level of the robotic platform, whereas the horizontal axis represents the day-time period (every dot point in the lines is associated to a battery-level measurement taken every 15-minutes). There are three lines in the figure corresponding to the three studied offloading policies: **i)** no offload (green line), **ii)** offload on the embedded ARM core of the FPGA (blue line), and **iii)** offload on dedicated low-level hardware blocks in the FPGA (red line). Finally, in all cases, the sharp ramp-ups indicate the battery charging periods.

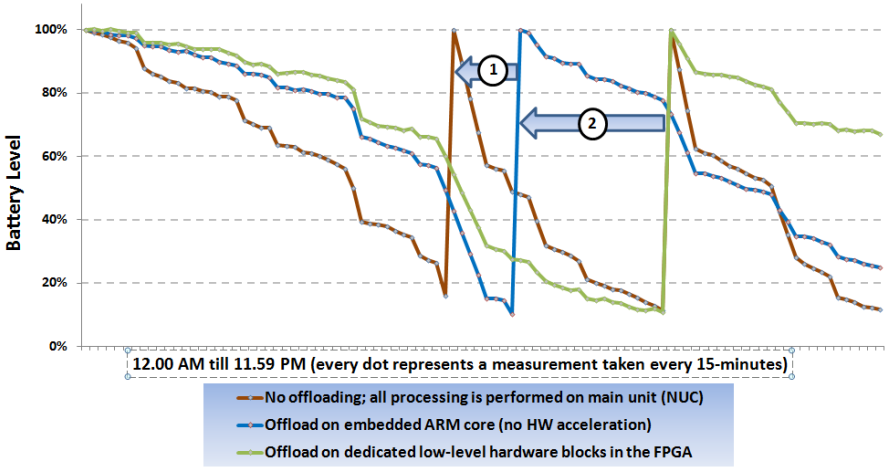

Figure 14 Run-time depletion of robot battery and number of required charges for the three studied offloading policies

As Figure 14 indicates, our offloading policies are able to significantly increase the autonomy of the robot. In our setup, the time required to charge the battery (from depletion to full capacity) is a 2-hours period. As a result, in the "no-offloading" case, for a time-window equal to 4-hours, the robot is not able to operate, thus it cannot follow the person to another room or most importantly it might miss capturing important data that are relevant to a critical situation or emergency. In addition, the charging periods coincide with periods of increased activity (as indicated by the results presented in Figure 12). On the contrary, our offloading policies (e.g., when the wake-up decision logic is implemented in the FPGA) manage to reduce the number of the required charges to one and to actually move the charging period to a time-slot of reduced activity.

In the next section, we present our methodology to assess the autonomy of the robot for a given target elderly or disable person and a given battery capacity.

5.2 Autonomy Estimator for AAL Robots

Having analyzed the strength of our profiling-based approach, we now present a practical, system level approach to leverage the profiling analysis results. The goal is to quantify the battery resources of the robot based on the profiling results of the person that is being monitored. To this end, Figure 15 depicts a high-level representation of the proposed approach (realized in python code in our setup). The inputs in this module are:

i) the profiling of the daily activities of the target person, and
ii) a diagram of the department of the target person

The output is a text file containing a set of robot autonomy-related parameters in the following format:

$$\texttt{Battery } \{C_i,\ Ch_i,\ NA_i\},$$

where:

- C_i is the capacity of the battery measured in mAH
- Ch_i is the number of required charges in a daily basis
- NA_i is the time period (measured in hours) that the robot is marked as non-available, thus it cannot perform its designated tasks

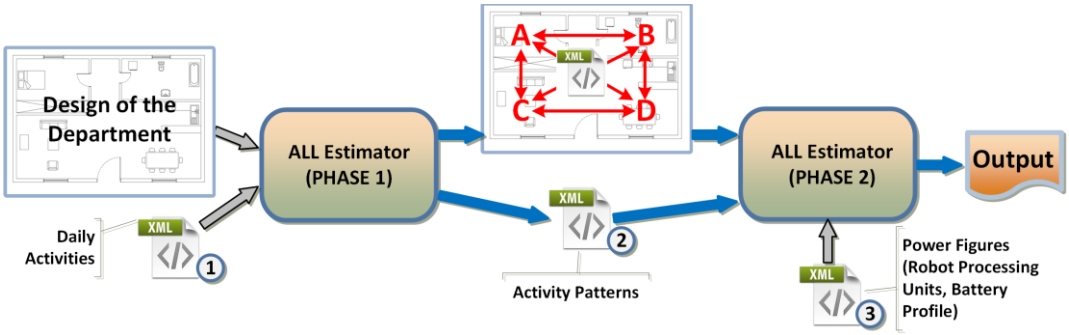

Figure 15 Data flow and main processing steps in the autonomy estimator

Note that *Ch* and *NA* are not identical (qualitatively) because it is possible to locate the charging station in a position that the robot is still able to provide useful feedback. In our analysis, we assume that the location of the charging station is predefined, and it is considered as input to the battery estimator (annotated in the design of the department). Finally, we also assume that only one charging dock exist in the house premises. It is worthwhile to mention that the previous assumptions are not mandatory. Our methodology can be easily extended to output one or more locations that would be more suitable (in terms of power reductions) to locate the charging station. However, this direction is not considered in this work.

5.2.1 Description of the Tool Main Stages

In the rest of this section, a description of the two main stages of the battery estimation tool is presented.

Stage 1: The data flow and processing steps of this stage are as follows:

[Input 1] Activity Patterns: This xml file (titled as *xml_1* in Figure 14) contains a description of the daily activities of the person being monitoring. In essence, this is a questionnaire filled by the personal care-givers. In the context of this work, the questionnaires were collected during the third pilot phase of the RADIO project. In particular, the care-giver recorded the activity patterns (e.g., watching TV, meal preparation etc) of the target person every 15-minutes. The questionnaires of multiple days can be collected and consolidated in order to end up with a more representative behavior of the target person. The activities of the targeted end-users are selected among a predefined set of daily activities generated with the help of the non-technical partners of the RADIO project.

[Input 2] Domestic Environment: The main purpose of this input is to capture the activity of the robot mechanical subsystem i.e., when the robot must follow the person to another room. This information might be provided in various formats, but in this work, we opted to represent this information in a simple xml format. Moreover, we assumed that that each room is a rectangle, so the room information can be easily formulated by two (x, y) pairs.

[Processing Phase]: In this phase, the previous two inputs are parsed and analyzed. The target is to extract specific statistical results that will represent the summary of the daily activities of the target person. The result of this processing step is two output files (in the form of xml files) that are described below.

[Output 1]: The design of the person's department is annotated with specific information based on the analysis of the target person activity patterns. In particular, the questionnaire filled by the personal care-giver is parsed in order to extract the specific locations within the house (bed, sofa, kitchen table etc.) that the target person performs a specific daily activity. The distances (marked with red arrows in Figure 14; the distances are measured in meters) between the annotated house locations (marked as "A," "B," "C," and "D") are then calculated. In case that there are multiple ways to move from a location "A" to a location "B," the shortest distance is considered (experimentally derived). The latter distance-related

information will be used in the next phase to account for the power consumed by the robot mechanical system.

[Output 2]: The second intermediate output is an xml file that contains the following information: **i)** how much time (measured in hours) the target person spent in a particular daily activity (e.g., watching tv, meal preparation etc) and the associated house location(s); the location information is needed to distinguish the case in which the robot is attached to the charging station, but it is still able to perform its designated tasks and **ii)** how many times the target person moved between the annotated locations included in the previous output file.

Stage 2: The two outputs of the previous stage are provided as inputs to the second stage of the AAL autonomy estimator. The data flow and processing steps of the second stage are:

[Additional Input] Power Figures: Consequently, this xml file (titled as *xml_3* in Figure 15) includes the following power-related information: **i)** the power consumed in each of the five states of the robotic subsystem (*with and without the proposed FPGA-based offloading mechanism*). The states are described in Section 4 and they are: *moving*, *monitoring*, *sensing*, *processing*, and *idle*; Note that the power figures of each discrete robot state are extracted by performing live, on-the-spot, measurements in our laboratory and **ii)** a model of the (dis)charging behavior of the robot battery (a linear model is assumed in this work).

[Processing Phase]: The main step of this phase is to associate (using a lookup table) each daily activity (included in the second intermediate output file) to one or more states of the robotic platform. The latter association is performed a priori (once for each robotic platform) and it is the result of the cooperation among the technical and non-technical partners of the RADIO project. As noted, a specific human activity (e.g., having lunch) might combine more than one of the robotic states (e.g., sensing, and processing).

Having the time spent in each daily activity (included in the second intermediate output file) and the power figure of each robotic state (included in the additional input of the second stage), then the battery resources consumed during the day can be estimated. Finally, it should be mentioned that the power consumed by the robot mechanical subsystem (part of the moving state of the robotic platform) is calculated by taking into account, the distance-related information captured in the first intermediate output file.

[Final Output]: The final output (rightmost part in Figure 15) is a cvs file targeting to quantify the autonomy of the robot for a range of realistic battery capacity levels assuming the person profile shown in Figure 13. An example screenshot is shown below:

```
Capacity  |  Offload  |  #of Charges  |   NA
2400mAH   |     NO    |       5.54    | 9.42
2400mAH   |    YES    |       2.79    | 3.91
                      ...
4800mAH   |     NO    |       2.79    | 3.91
4800mAH   |    YES    |       1.74    | 2.27
                      ...
9600mAH   |     NO    |       0.97    | 1.16
9600mAH   |    YES    |       0.68    | 0.82
                      ...
```

Figure 16 Estimator output

The first column in the above screenshot depicts the studied battery capacity levels while the second column shows if the HW FPGA-based acceleration mechanism is utilized. The third and fourth columns illustrate the number of the required battery charges and the NA parameter (robot non-available; NA is measured in hours) in a daily basis. As the screenshot indicates, our offloading technique is able to decrease significantly the NA parameter in all battery levels. As expected, the impact of our offloading technique is more pronounced in lower battery capacities.

The output of our tool can be used by the care-givers in order to end-up with safe conclusions regarding the required battery (thus the autonomy) of the AAL robot. As a result, the burden of constant monitoring (by a third person) of the elderly or disable person can be reduced (to the extent possible).

6 ACCELERATING IMAGE PROCESSING ALGORITHMS

The algorithm for monitoring the state of the patient is based on center of gravity calculation and can be divided into 4 to 5 parts, depending on whether mark ups are activated or not. Figure 17 shows the general functionality of the algorithm as schematic and as pseudocode.

1. Reading of the most recent image frame:
   The image data is provided by the Asus Xtion Pro camera of the RADIO robot platform. The image data is sent via USB directly to the NUC which publishes the received frames via its robot operating system to the Avnet Picozed where it is processed. The image frame is then read by the software and saved to a 3-dimensional array. The first two dimensions indicate the pixels positions whereas the third dimension stores the color values of the RGB color channel. Each color is coded with 8 bit, resulting 24 bit color payload. Given that the Asus Xtion Pro camera provides images with the size of 640×480 pixels, the resulting array size is 640×480·3 = 921600 or 900 KiB.
2. Detection of movement:
   The algorithm loads to subsequent frames and compares both image frames with each other in order to detect changes or movement within the two image frames. In order to reduce the impact of small movements of the camera or image noise, the comparison does not only take place on the subtracted image, but rather on blocks of pixels with the size 10×10. Within these blocks the mean value of all subtracted color channels is calculated. If this value exceeds a certain threshold, the respective block is flagged as active to show that a change has occurred. While the person is moving out of the bed, the pixel blocks that detect movement are highlighted in red.
3. Calculation of center of gravity:
   After all blocks have either been detected as active or inactive, the center of gravity can be calculated. In this case, the center of gravity is calculated through the mean value the positions of all active blocks. Because the active blocks are positioned in the middle of the image and in the lower right corner of the image, the center of gravity lies directly in the between the detected hotspots of movement.
4. Evaluation of center of gravity:
   Now that the position of the center of gravity has been determined, its position needs to be analyzed and interpreted. If the y coordinate of the center of gravity exceeds a certain threshold, the algorithm assumes that the observed person has gotten out of bed. Several of these thresholds exist.
5. Drawing mark ups:
   In order to optimize and help debug the algorithm, markups can be drawn into the image. When drawing markups, all color values which differ more than the value 40 compared to the prior frame are set to 70. If the pixels differ less than 40, the color values are quartered. Additionally, the pixel within an active block will be colored red. This is done by adding the value 128 to the red channel. This calculation is saturated, meaning that the resulting value never exceeds 255.

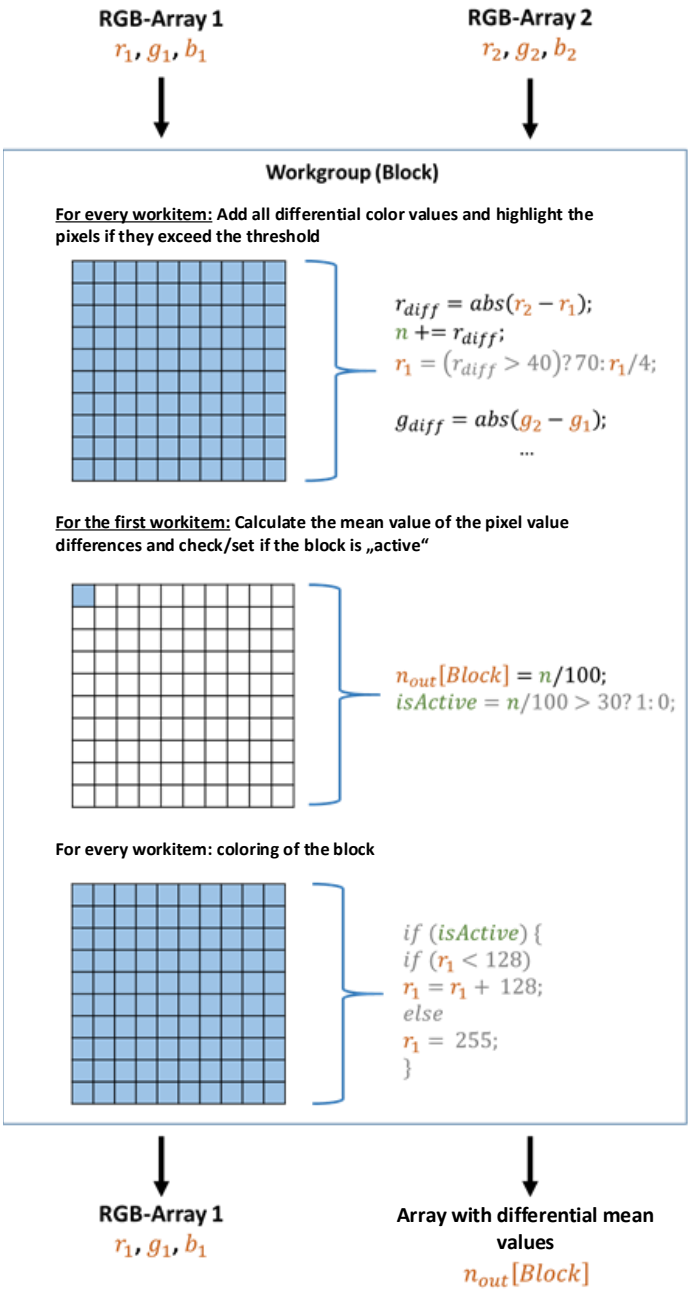

Figure 17 Schematic view of the kernel design annotated with pseudocode

6.1.1 Profiling Results

In order to optimally accelerate the image processing algorithm with programmable hardware, the compute intensive components need to be identified. This is done with the help of profiling. The Picozed is a System on Chip with an dual core ARM Cortex A9 processor and integrated programmable hardware. The image processing algorithm is first executed on the ARM processor. There, the performance of the algorithm is determined and the potential hardware accelerated components are identified. From the software side, the algorithm consists of several subblocks which are further analyzed during the profiling. These are described in Table 8.

Table 8 Subblocks of the algorithm

Profiled functions of the algorithm	
Function name	**Task**
copyToRGB	Copy the received image data to a 3-dimensional array
checkBoxes	Calculate the mean value of the color value differences over the last 2 frames and indicate the active blocks.
annotateBoxes	If markups are activated, indicate the pixel changes and highlight the active blocks.
process_function	Calculates the center of gravity and determines its position. This is the function that calls *checkBoxes* and *annotateBoxes*.
copyToImageData	Copy the processed image data from the 3-dimensional array to a ROS compatible array for debug purposes.

For each profiling run, the algorithm is executed 20 times in order to mitigate the impact of outliers. The used profiler is gprof and the results are presented in Figure 18 for the algorithm with markups and in Figure 19 for the algorithm without markups.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
61.70	2.32	2.32	61440	0.04	0.04	checkBoxes
17.02	2.96	0.64	20	32.00	32.00	copyToImageData
16.49	3.58	0.62	20	31.00	31.00	copyToRGB
3.72	3.72	0.14	13199	0.01	0.01	annotateBoxes
1.06	3.76	0.04	20	2.00	125.00	process_function

Figure 18 Profiling results of the algortihm with activated markups

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
51.29	1.39	1.39	61440	0.02	0.02	checkBoxes
26.57	2.11	0.72	20	36.00	36.00	copyToImageData
21.03	2.68	0.57	20	28.50	28.50	copyToRGB
0.74	2.70	0.02	20	1.00	70.50	process function

Figure 19 Profiling results of the algorithm without activated markups

As can be seen in both Figures, the algorithm spends most of total processing time in the *checkBoxes* function. In the case with activated markups, the amount is 61.70% and 51.20% without activated markups. Because the *copyToImageData* function is only required for debug purposes, this function will not be implemented in the final algorithm design. Therefore, the timing value for this function is ignored. In the case of activated markups, all the data required for the *annotateBoxes* function is generated by the *checkBoxes* function. Because both functions are executed sequentially, it is possible to generate hardware accelerators for both functions.

6.1.2 Hardware Accelerator Design

In order to efficiently switch between the algorithm with and without markup functionality, two OpenCL kernels are designed. This enables an efficient implementation of only one query in order to determine which kernel version will be executed. OpenCL kernels consist of workgroups and workitems. In this case, a workgroup stands for one pixel block and a workitem stands for one pixel. The designed kernel will then be called 640×480=307200 times for each pixel pair. The first step is to calculate the

difference of all color vlaues of each pixel pair in a workgroup. If the differential value exceeds the value 40, the pixel value is set to 70, otherwise the value is divided by 4. As soon as each workitem of a workgroup completes the differential calculation, the first workitem of the workgroup will calculate the mean value of all workitems. The mean value is then saved to an external array which is accessible by the CPU for further processing. If the mean value exceeds the threshold value of 30, the block will be highlighted in red. Figure 20 shows the kernel implementation as schematic and as pseudocode.

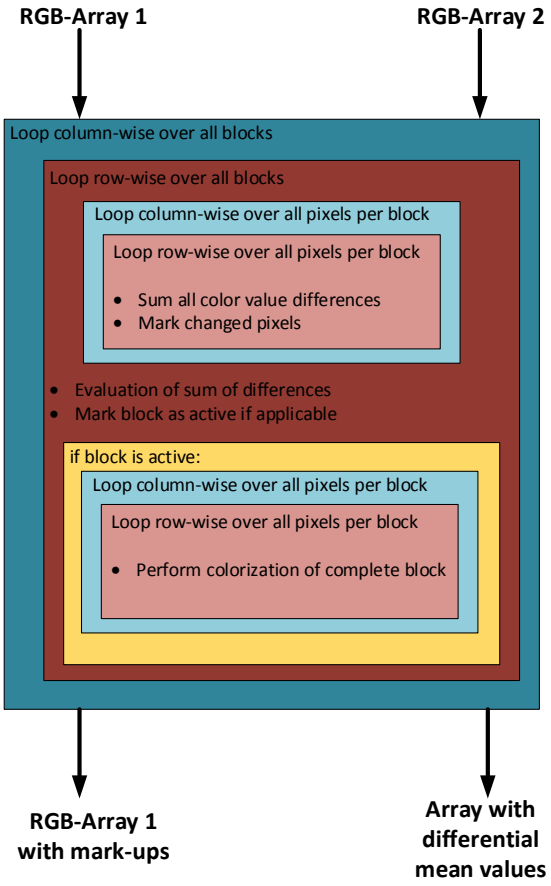

Figure 20 Depiction of the implemented algorithm

The initial version of the OpenCL code can be generated with 100 MHz. Figure 21 shows the resource requirements of the initial hardware version. This core is compared to a software implementation on the dual core processor of the Picozed. The execution time of the algorithm on software takes approximately 17547 µs. The generated hardware requires 88404 µs, meaning the hardware accelerator requires 88404 µs · 100 MHz= 8840400 cycles to execute the algorithm. This results in a speedup of 0.2. In order to achieve an accelerator which actually accelerates the image processing algorithm, further optimization steps have to be executed.

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1827
FIFO	-	-	-	-
Instance	2	4	662	812
Memory	-	-	-	-
Multiplexer	-	-	-	2599
Register	-	-	2160	-
Total	2	4	2822	5238
Available	280	220	106400	53200
Utilization (%)	~0	1	2	9

Figure 21 Resource uitilization of the initial OpenCL kernel

The first optimization step is to efficiently let the accelerator read the image data from the DDR memory. This is done with the command *async_work_group_copy*. This command transmits a user defined number of sequential bytes from memory via a burstmode to the accelerator. The transmission of one frame is executed stepwise in order to reduce the resource usage of the BRAM on the programmable hardware. Because one image always lies sequentially in memory, only one transmission command per frame is required. After this step, the estimate cycles to complete the algorithm are in a range from 4729607 - 5712647 cycles, which means a performance improvement of 36% - 46% compared to the initial implementation. This performance improvement however comes at the cost of an increased resource utilization as can be seen in Figure 22. Here, the number of used BRAM blocks has increased from 2 to 74 while all other resource remain almost constant.

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1591
FIFO	-	-	-	-
Instance	2	4	662	812
Memory	72	-	0	0
Multiplexer	-	-	-	1340
Register	-	-	1786	150
Total	74	4	2448	3893
Available	280	220	106400	53200
Utilization (%)	26	1	2	7

Figure 22 Resource utilization of the OpenCL kernel after optimizing the data access

Because the number of required BRAMs is very high, the memory requirements of the accelerator are reduced in the second optimization step. Currently, every color value is transmitted as a 4 Byte value to the BRAMs although a 1 Byte value would suffice. Therefore, all three color values are stored in one 4 Byte value on the software side and then transmitted to the accelerator. This reduces the data transmission by 2/3 from 14535 cycles to 4935 cycles. By performing this optimization, the performance of the accelerator is increased while also reducing the resource utilization. This is shown in Figure 23. The number of BRAMs is reduced from 74 to 42 and the LUT resource utilization is reduced by 2% compared to the first optimization. The estimated cycle number is also further reduced to 2272007 - 2947847 cycles which is an performance improvement of 52% compared to the first optimization step.

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	691
FIFO	-	-	-	-
Instance	2	4	662	812
Memory	40	-	0	0
Multiplexer	-	-	-	1245
Register	-	-	1383	140
Total	42	4	2045	2888
Available	280	220	106400	53200
Utilization (%)	15	1	1	5

Figure 23 Reosurce utilization of the OpenCl kernel after optimizng memory requirements

Since image processing algorithms perform many operations on each pixel individually, these operation are executed in a loop. These loops can be parallelized on hardware. Parallelizing a loop can be done through loop pipelining or through loop unrolling. While loop pipelining reuses the already available components for parallelization, loop unrolling requires separate resources in order to increase the degree of parallelism. Therefore, loop pipelining requires less additional resources than loop unrolling. The algorithm has 5 loops that can benefit from either loop unrolling or loop pipelining, see Figure 20. In the case of this algorithm, no performance difference is detected when using loop unrolling compared to loop pipelining. Because loop pipelining requires less hardware resources, loop pipelining is used for 2 of the 5 loops. In the other 3 loops, no performance improvement was measured when employing pipelining or unrolling techniques. Figure 24 shows the resource utilization when employing loop pipelining for the algorithm. Through loop pipelining, the resource requirements of the BRAMs are reduced even further from 42 to 38. The number of DSP blocks is also reduced from 4 to 1 and the FFs are slightly increased as well as the LUT resource usage. This optimization further increased the performance compared to the last optimization step, leading to cycle number of 1273607 - 1586951 which is an acceleration of 44%-46%.

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	2142
FIFO	-	-	-	-
Instance	2	-	662	812
Memory	36	-	0	0
Multiplexer	-	-	-	1344
Register	-	-	1927	140
Total	38	1	2589	4438
Available	280	220	106400	53200
Utilization (%)	13	~0	2	8

Figure 24 Resource utilization of the OpenCL kernel after optimizing loop executions

After these three optimization steps, the accelerator is again compared to the software implementation of the algorithm.

6.1.3 Evaluation

In order to evaluate the performance of the accelerator on the real hardware, the accelerator must first be implemented on the PicoZed platform. This is done with the Vivado tool provided by Xilinx. The accelerator must be connected to the processing system in order to receive the image data from the DDR memory. Table 9 shows the execution times of the different implementation versions. For all implementations, the clock frequency of 100 MHz is used. The ARM processor is running at 666 MHz.

It can be seen that the initial and up until the second optimization hardware version, the software version outperforms the hardware implementation. This changes in the third optimization where the hardware implementation reaches a speedup of 1.32 compared to the software version. All hardware implementations can further increase their performance compared to the software implementation by increasing the clock frequency.

Table 9: Measured execution times of each optimization step and of the software implementation

Execution times and Speedup compared to the Software implementation		
Measurement platform	**Execution time**	**Speedup**
Software (ARM)	17547 µs	1
Initial Implementation	88404 µs	0.2
First optimization	48687 µs	0.37
Second Optimization	23401 µs	0.75
Third optimization	13290 µs	1.32

7 MANAGING THE RADIO COMPUTATION PLATFORM USING SOFTWARE ANALYSIS TOOLS

This chapter describes the usage of several profiling tools in order to optimize the RADIO computation platform.

7.1 Summary of D4.1 and D4.2

In deliverable D4.2, a proof of concept scenario for the hardware software co-design was presented using the discrete cosine transform (DCT) kernel. This kernel consists of the 2D transformation and the transpose calculation. These two parts exhibit vastly different behaviour when implemented on the ARM and on the FPGA, see Table 10.

The performance of the 2D transformation and of the transpose calculation can be improved in both cases. Depending on the optimization focus, a large design space has to be explored. Xilinx' SDSoC is a framework that supports this type of design space exploration. Therefore, this framework is being exploited for further use in the RADIO project.

7.2 Designing a System of Distributed ROS Nodes

rostune is a tool that helps ROS developers distribute their nodes in the most effective way. It collects and visualizes statistics for topics and nodes, such as CPU usage and network usage. rostune was specifically developed for RADIO, to allow us to experiment (at development time) with the optimal way to distribute nodes between the robot's on-board computer and the computational units available at the home. This need appears in indoors home assistance or industrial scenarios with good connectivity and easy access to on-site computing units, where off-board computations can limit battery consumption.

The factors involved in this decision are the CPU and memory requirements of each node, bandwidth requirements of each topic, and sensitivity to dropped frames. The visualization of these statistics assists in understanding the dynamics of the system and of the exchange of messages between nodes, so that ROS developers can make informed decisions about how to best distribute the ROS nodes between the available processing units. rostune collects statistics with a minimal CPU, memory, and network usage footprint of its own, it operates in both single-core and multi-core distributed ROS systems, and results are collected and visualized in PlotJuggler, the Qt based application that visualizes ROS message streams as a time series.[4]

Table 10 Latency and power profiling of DCT's taskss

	CPU	FPGA
2D transformation	1.93 ms	0.45 ms
Transpose calculation	0.09 ms	0.86 ms

[4] rostune does not have a hard dependency on any particular visuzalization tool, but its output format is compatible with PlotJuggler, http://www.ros.org/news/2017/01/new-package-plotjuggler.html

Laser scan analysis (blue plot) for tracking moving people can be efficiently executed off-board due its large CPU requirements but small bandwidth footprint.

Figure 25: A characteristic example of CPU time (left) and bandwidth (right) usage.

This decision is based on prior requirements (such as the need to keep critical obstacle-avoidance nodes on-board) but is also influenced by more dynamic considerations such as typical CPU and bandwidth usage. Figure 25 shows a characteristic example from RADIO experiments: the CPU-intensive pattern recognition algorithm for the *4m walking ADL* can be efficiently executed off-board due its small bandwidth footprint. This rather pronounced example could have been easily guessed, but there are also subtler architectural decisions. Vision algorithms, for example, are both CPU and bandwidth-intensive so it is not easy to decide without empirical evidence where along the vision processing pipeline is a good point for transferring the processing off-board.

8 THE RADIO MAIN CONTROLLER

8.1 Architecture

The RADIO Main Controller is the main orchestrator of the behaviours of the RADIO Home and the main keeper of the information collected and analysed by the various RADIO Home systems. Its functionalities include:

- System orchestration
- Bridging between the different sub-systems
- Storing and serving ADL recognition results

The Main Controller is (physically) partially distributed between the home computer and the robot computer, via a dual-ROS core architecture. This adds integration complexity compared to the earlier single-ROS core RADIO architecture, but address the problem that:

- The Main Controller would be unable to operate with the robot turned off or having run out of battery, if the robot's computer executed the only ROS core process in the system.
- The bandwidth-hungry communication channels between the sensors and the perception modules would have to use the wifi, if the home computer executed the only ROS core process in the system.
- The Main Controller would be unable to operate with the robot turned off or having run out of battery, if the robot's computer executed the only ROS core process in the system.
- The bandwidth-hungry communication channels between the sensors and the perception modules would have to use the wifi, if the home computer executed the only ROS core process in the system.

8.2 Orchestration

The action and node manager orchestrates the overall system, including reacting to user initiatives through the user device and initiating automated actions, except for home automation directly handled by the S&C suite.[5] Orchestration is implemented by sending control messages and by switching the state of the perception and bridging nodes between "active" and "idle". Idle nodes consume practically no CPU resources (cf. Section 9.1), so the starting/stopping functionality was deprecated. Similarly, the mechanism for monitoring ROS node execution has been converted to also use the state-reporting services.

Action and node management is distributed between two nodes:

- The main node that executes at the home computer: https://github.com/RADIO-PROJECT-EU/radio_node_manager_main_controller
- The robot-side node that executes at the robot's on-board computer: https://github.com/RADIO-PROJECT-EU/radio_node_manager

The main node relegates to the robot the distribution of control messages for the ROS nodes executing on the robot. Only the main node is required for the operation of the overall RADIO Home, so that functionalities not related to the robot remain active even if the robot is off-line or turned off.

8.3 ZWave and MQTT Network Bridges

The Main Controller bridges between the ROS middleware/wifi network and two other communication infrastructures present in the RADIO Home:

[5] The cloud-based S&C rule engine that implements pre-configured automations and the EnControl GUI for monitoring sensors, see also D5.5 User Interfaces.

- The REST API to the ZWave network of home automation sensors and actuators, via the S&C Gateway: https://github.com/RADIO-PROJECT-EU/snc_sensors_publisher
- The MQTT middleware used by the BLE network: https://github.com/RADIO-PROJECT-EU/room_status_publisher

These components bridge between networks by simultaneously being REST/ROS client and MQTT/ROS client, respectively.

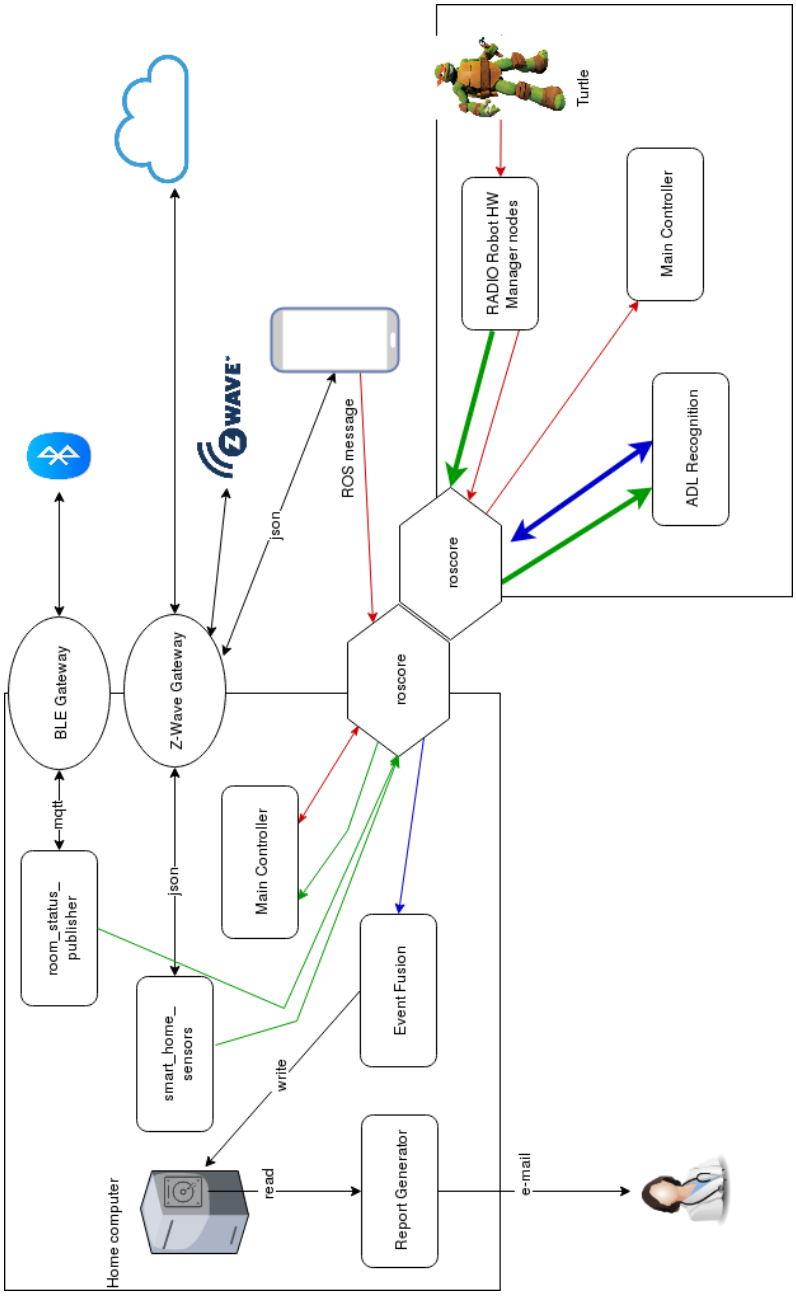

Figure 26. Interconnections between the Main Controller, Turtlebot, and the home automation components.

8.4 ADL Recognition Wrappers and Report Generator

The ADL Wrappers are a collection of ROS nodes that are aware of the RADIO Home database schema and of the semantics of the ROS messages published by the ADL recognition methods. These wrappers listen to the ADL recognition methods and make, where necessary, calculations such as extracting a duration from an event marking the start of an ADL and the matching event marking the completion of the ADL. These wrappers output to a temporary, short-term database. This database is used by the Report Generator to compute the daily or other aggregations that need to be reported and stored in the long-term database.

- Wrapper for walking pattern recognition in rage data (D3.4, Section 2): https://github.com/RADIO-PROJECT-EU/hpr_wrapper
- Wrapper for visual recognition of motion events (D3.4, Section 4): https://github.com/RADIO-PROJECT-EU/motion_analysis_wrapper
- Wrapper for moving object tracking (D3.4, Section 3) and classification (D3.5, Section 2): https://github.com/RADIO-PROJECT-EU/ros_visual_wrapper
- Wrapper for composite events that combine sensing across different networks https://github.com/RADIO-PROJECT-EU/snc_events_wrapper
- Report Generator: https://github.com/RADIO-PROJECT-EU/radio_report_generator

Similar wrappers will also be developed for the acoustic event recognition method (D3.5, Section 3) and for the rules that extract events from the home automation sensors (D3.5, Section 4).

Table 5: Access levels and authentication for the RADIO Home database

Component	Access Level	Authentication	Explanation
RADIO Home components	Write access	Only accessible from the internal RADIO Home network	The RADIO Home components that recognize events update the event log.
Report Generator	Read access	SSL-based authentication.	Read access for the formal caregiver of this specific home, using conventional authentication and access control mechanisms.
Notification Generator	Read access	Only accessible from the internal RADIO Home network	Filters the data for events that trigger notifications.
RASSP	Read access	Only accessible from the internal RADIO Home network	The RADIO privacy-preserving data mining component accesses all data to respond to queries that observe the RASSP Protocol (cf. D5.6). Access through RASSP guarantees that these responses allow statistical aggregates to be computed over many RADIO Homes without revealing the values of any one of these Homes.

8.5 Data Services

The main requirements that must be satisfied by the technologies used originate from the nature of the stored data and the nature of the consumers of those data. The output of the analysis algorithms (cf. Section 5.2, D3.3 *Conceptual Architecture*) is the log of the recognized events annotated with the type of the event, the actual time and date that the event occurred, and the duration or other measurement associated with the event, if any.

Since the recognized events are recurrent this log forms essentially a set of time-series for each event type. A time-series database is a database that is optimized for handling time series data, promoting time as a first-class citizen and implement time-based operations in a more efficient way.

This database needs to provide access as foreseen in Table 5.

We used the InfluxDB database management system,[6] an open source scalable time-series database that targets use cases that heavily use time-based metrics and sensor data in the IoT context. The current RADIO data schema contains a *measurement* (i.e., a database table) that includes all the higher-level events produced by the RADIO Home analysis algorithms. This measurement has the following fields:

- **event_type:** the type of the event recognized, as tagged by the recognition algorithms, such as “4m-walking”, “Sitting-to-Standing”
- **time:** the timestamp at which the event was recorded
- **duration:** the duration of the event, if applicable

[6] Cf. https://www.influxdata.com

9 ROBOT BEHAVIOUR

9.1 Task Switching

The action and node manager orchestrates the overall system, including reacting to user initiatives through the user device and initiating automated actions, except for home automation directly handled by the S&C suite.

Orchestration was previously implemented by sending control messages and by starting and stopping ROS nodes. For the final prototype, all orchestration is carried out by setting nodes in and out of an "idle" state, where they are immediately available (i.e., the node process is executing) but they do not consume and process any messages, they do not publish any message, and only use minimal processing and network resources.

9.1.1 Motivation and Requirements

The orchestration system consists of two major parts:

1. The communication system between the node manager and the rest of the ADL-related nodes that run in different machines (main controller - robot equivalently).
2. The methodology to start and stop each ADL based on a variety of scenarios.

Communication:

The communication inside the RADIO system was mainly based on ROS topics. ROS topics are code-named channels that contain specific content. Many nodes (processes) can subscribe (listen) to the same topic and also many nodes can publish (transmit data) to a topic. This architecture makes ROS topics very easy to use, since anyone can connect to a specific channel and then acquire or transmit information.

Since ROS topics provide a many-to-many type of connection, they are more appropriate for data streams, disregarding their ease of use. In our node handling scenario, what we really needed was a way to send a start/stop signal from the main controller to the robot, to control the desired ADL related node. Thus, what was really needed was a one-to-one communication. In an earlier version of the RADIO orchestrator, the different nodes that made up the distributed Main Controller (Raspberry node, robot node) communicated over a control topic. ROS services provide a one-to-one connection, in which return data is possible. This provides a request-response protocol just like calling a method from within the code.

Using ROS services enables us not only to safely transmit data to another node, but also get an answer from them, ensuring the normal flow of the procedure. Of course, requiring a response after each service call adds to the complexity of the system, but it is tolerable for the sake of robustness.

Starting/Stopping ADL nodes:

The management of ADL related nodes was based on starting and stopping their processes. This included at least two processes for each ADL, one for the processing and one for the consumer of its results (wrapper). In more extreme cases, like the 4-meter ADL, five processes had to be started/stopped in total. Managing the flow of the system by starting and stopping nodes is neither the most elegant nor the most efficient way. The problem of elegancy is pretty obvious: processes frequently starting and stopping keeping the operating system busy, managing all those changes.

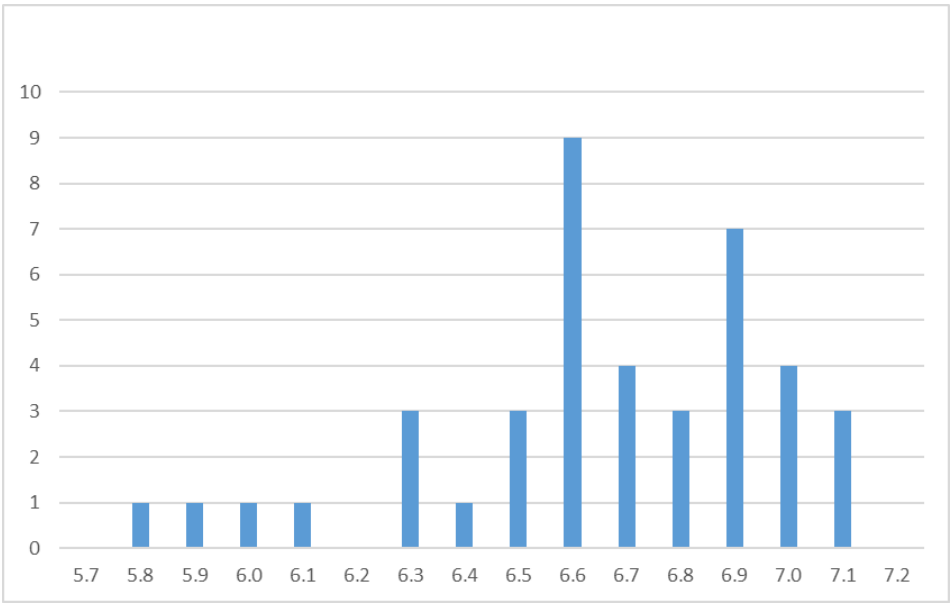

Figure 27: Time (in sec) to load executable, start the process, and connect to the ROS middleware.

Notes: Time measured for a relatively small and simple system of two nodes (joystick teleoperation receiver and velocity smoother), available here: https://github.com/RADIO-PROJECT-EU/turtlebot/tree/master/turtlebot_teleop

Measurements are made on a NUC Intel Celeron @ 1.6GHz, 2GB memory, executables loaded from SSD, Ubuntu 14.04, ROS Indigo, only executing the OS, the ROS Master, and one other node that generates traffic for the experiment node to subscribe to. Measurements refer to the time from invoking node execution until the node has connected the ROS master, i.e., time includes process setup time, time to establish network socket to the ROS master, and time to register as a subscriber to a topic.

The efficiency problem lies not only on the added operating system activity, but also on the fact the when a new ROS node starts, it takes approximately three to eight seconds for it to get registered to the master and initialize its subscribers (Figure 27).

Utilizing the power of ROS Services, the latest version of the orchestration system was developed. In this version, all ADL related nodes are always running and offer a service that can alter their state based on received data. We will thoroughly discuss the architecture of this implementation in the next section.

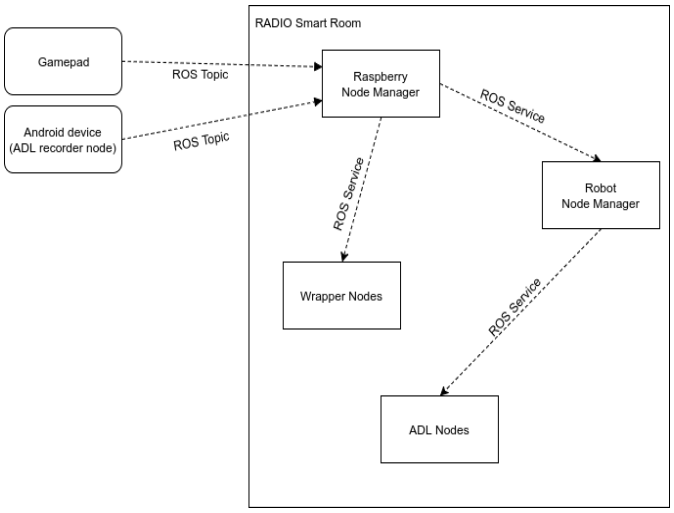

Figure 28: Orchestration system

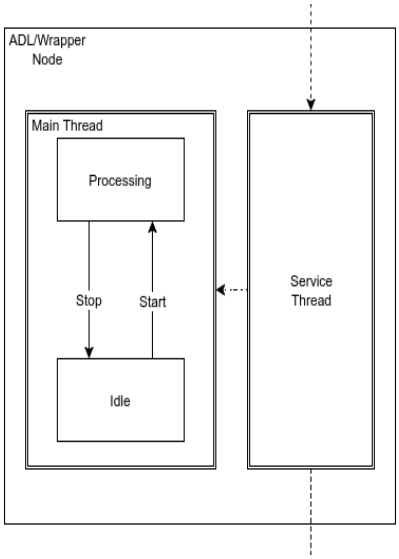

Figure 29: Node architecture

9.1.2 Architecture

Figure 28 shows an abstract version of the current orchestration system. The Main Controller listens to commands from user devices like gamepads and Android mobile phones and then informs the corresponding nodes using ROS services. More specifically, the Raspberry Main Controller invokes the Robot Instruction Receiver Service to relay to the robot's Main Controller the ADL code that needs to start or stop while simultaneously invokes the corresponding ADL wrapper's Node State Service. When the robot receives the message from the Main Controller, it also sends a message to the selected ADL node with the desired state. All these messages between the Main Controller, the Robot and the ADL related nodes along with their wrappers, are sent using ROS services. Each time, the input data is the desired action, and the return value is the result of the requested process.

Figure 29 shows an abstract architecture of a node that is involved in the newly implemented ROS service based state selection. In more detail, and as shown in the diagram, the node offers a service that receives as an input the desired state, and returns its current state. The possibility to just request the current state of each node, without altering its state has also been implemented. For example, a call on a State Service with the generic type <Service Type> in Python, would look like this:

```
command = 1
state_service = rospy.ServiceProxy('hpr_wrapper/node_state_service', <ServiceType> )
new_state = service(command)
```

In the example above, the service 'hpr_wrapper/node_state_service' is the one provided by the 4-meter walk ADL wrapper. The command sent is the number "1" which then is translated in the service callback as a "Start message", and enables the wrapper's processing. Alternative command values for all the wrapper nodes include "0" for a "Stop Message" and "-1" for no state change. All three alternatives receive as an answer the current state of the node (running/idle).

Specifically, for ADL recording, some nodes apart from the desired state, also receive the ADL code name and repetition. These extra parameters help in making the reports include human provided codenames that can distinguish multiple recordings of the same ADL.

9.1.3 Implementation

The above has been implemented in the main controller (both Raspberry-side and robot-side node) and in all ADL processing nodes and their wrappers:

Package name and description	Source code repository and release implementing task switching	
Node manager: The home computer-side node of the Node Manager.	https://github.com/radio-project-eu/radio_node_manager_main_controller	v2.0
Node manager: The robot-side node of the Node Manager.	https://github.com/radio-project-eu/radio_node_manager	v2.0
HumanPatterRecognition: Recognizes human walking patterns in laser scans and tracks walking.	https://github.com/radio-project-eu/HumanPatternRecognition	v3.0.0
HPR Wrapper: Uses HPR output to recognize and time "walked 4m" events.	https://github.com/radio-project-eu/hpr_wrapper	v2.0

ROSVisual: Tracks moving objects in the RGB/depth modality and classifies motion as bed or chair transfer.	https://github.com/radio-project-eu/ros_visual	v2.0
ROSVisual Wrapper: Uses the output from ROSVisual to time chair and bed transfer events and to recognize and time "walked 4m" events.	https://github.com/radio-project-eu/ros_visual_wrapper	v2.0
Motion Analysis: Recognizes motion and classifies it as "bed transfer" and "pill intake" events.	https://github.com/radio-project-eu/motion_analysis	v2.0
Motion Analysis Wrapper: Uses the output from motion analysis to time the bed transfer event.	https://github.com/radio-project-eu/motion_analysis_wrapper	v2.0
Presence events: Uses the events published by the ZWave/ROS bridge, to log ADLs (TV watching, cooking, and presence events) inferred from the presence sensors and appliance usage sensors.	https://github.com/radio-project-eu/snc_events_wrapper	v1.0
Report generator: A ROS node that generates medical reports based on the information created by the wrappers.	https://github.com/radio-project-eu/radio_report_generator	v1.0

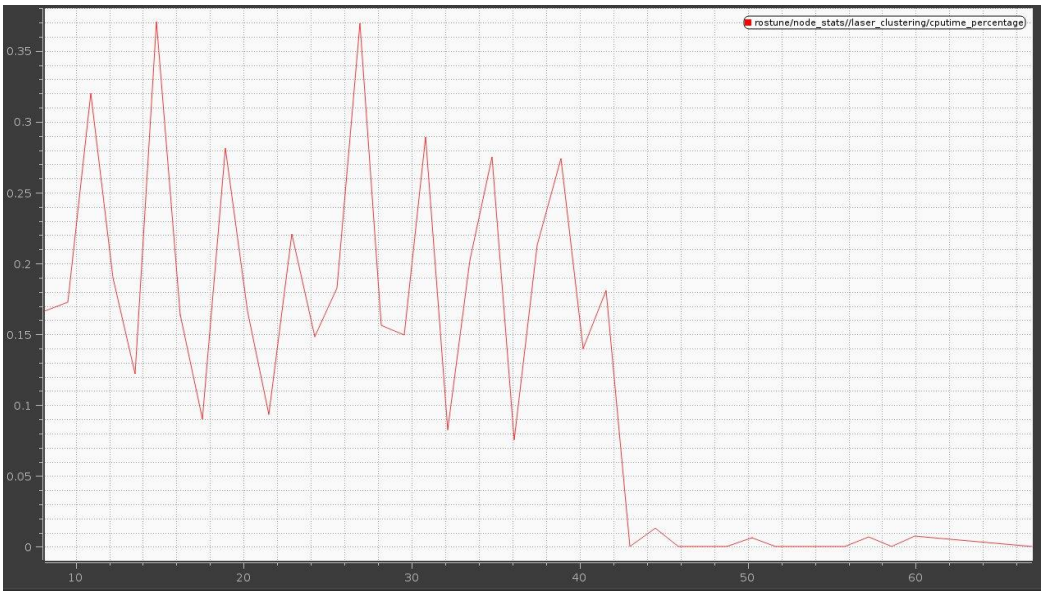

Figure 30: CPU usage of one of the four nodes that are responsible for the 4-meter walk ADL, measurements made using rostune

Generally, for all the nodes complete understanding of their internal data structure was needed, in order to distinguish which of the values needed to re-initialize after each change of state, and which needed to stay unmodified. The following nodes needed deeper adaptation than a simple implementation of the state changing architecture:

4-meter walk ADL:

This ADL's method consists of four ROS nodes that are connected using a linked chain model. These nodes were at first designed to run specific tasks only during their initialization, so those parts of the code had to be modified in order to run each time the nodes come on the "running" state. Since those nodes are connected in a linked chain model, the first node in the chain has been given the ability to enable all other three nodes, by using their provided state service. functionality of starting and pausing their processing methods was much more complicated.

Bed Transfer ADL and Pill Intake ADL

The methods from these two ADLs are very sensitive to image changes, so their initialization should take that into account. When the camera driver is polled for an image, there is a small stutter that could cause the two ADLs to produce false results. Based on this observation, the two nodes first enter a "semi-running" state, in which they are subscribed to the image topic, but do not produce results. Although this process would not require more than one second, due to the possible network delay that could occur in a clutter network, there is a five second gap between the transition from "semi-running" to the "running" state.

9.1.4 Measurements

Figure 30 shows the CPU usage of one of the four nodes that are responsible for the 4-meter walk ADL. At first, the node is in "running" state, which means that it has already received a message via its state service. At approximately the 43rd second, the node received another message which made it set its state to idle. This is a representative example of how all the other nodes behave, and also how few CPU resources are consumed when idle.

When idle, the nodes unsubscribe from all their subscribed topics, completely nullifying network usage.

9.2 Navigation in Cluttered Spaces

At the First Integrated Robotic Platform (D4.6), robot navigation was based on the standard parameters and configuration for navigation and obstacle avoidance. The following situation was occasionally observed during the first round of pilots at FHAG: the robot would remember that a corridor was congested with people and refuse to navigate to a goal that required passing through that corridor, even after the congestion has cleared.

To address this, we added the provision that if the goal cannot be reached, previously discovered obstacles that are currently not visible are removed from the costmap and the costmap is re-initialzed from the static map, forcing the robot to double-check if the obstacles persist. The robot will only give up after trying twice.

10 CONCLUSIONS

This deliverable presents the physical architecture of the RADIO Home, covering RADIO device interconnection and interfacing, specifications on interfacing the different domains, and on fast and energy efficient data processing in the distributed RADIO environment. More specifically, this deliverable includes the design of the physical architecture of the RADIO Home, and especially the wireless communications architecture between the RADIO Robot platform, the Smart Home devices, and the Main Controller that make up each RADIO Home. Second, the design of the architecture and the policies for managing the heterogeneous computing elements of the RADIO Home, including the central server, FPGAs, and the on-board Robot controller. Special care was given in investigating the most efficient way, in terms of power and delay overhead, to process different kinds of sensor data in the distributed RADIO environment and in observing the privacy for the user.

With respect to communication substrate, the Robot interface is defined and implemented and the WiFi and BLE connectivity is verified. The Z-Wave devices are able to be accessed through the RESTful API in the Home Controller gateway. The backbone of the smart home architecture is the WIFI/LAN interconnection between the router, robot, and RADIO Home Controller gateways. The router enables the communication with the IoT Platform and the WIFI/LAN infrastructure enables the information exchange between each component.

In addition as part of this work, alternative hardware and sensor positioning configurations are also investigated as part of this task with the focus on power and performance trade-offs between fixed function accelerators and more programmable (or even pure software) solutions. The programmable solutions offer more flexibility to provide several dedicated services to the end-users through software updates or extensions. However, fixed logic hardware solutions offer the significant advantage of privacy for two main reasons: i) the sensors data are pre-processed and immediately destroyed and ii) in case that further processing is required, this is performed on anonymized data (the outcome of the pre-processing step).

Dedicated hardware components that include also special low power modes are implemented in the Picozed FPGA. In the low power mode, only the FPGA is active and performing periodically sensor update from the Python camera. The Python camera is directly connected to the FPGA hardware and therefore does not require an additional processor for transferring image data to the programmable logic. Therefore, all systems that are not required for camera usage can be put in sleep mode until the image processing core in the Python camera chain wakes up all other systems.

A profile-driven, system-level approach to increase the autonomy of the robotic platform in AAL environments is benchmarked in experimental conditions via use case profiling. Moreover, a systematic methodology (realized as python tool) that outputs the autonomy of the robot for a range of battery recourses is also described. The inputs in our methodology are the daily activity patterns of the target elderly or disable people, information about the domestic environment, and the power figures of the robotic platform (with and without HW FPGA-based offloading policies). The daily activity patterns were collected by care-givers personnel during the third pilot phase of the RADIO project. The proposed methodology is considered as a useful tool for estimating the required battery resources (consequently the cost since it represents a significant part of the overall cost) of an AAL domestic robot.

Finally, this report documents work on improving the behaviour of the robotic platform and its integration in the RADIO Home system. This includes improving the main controller, the main orchestrator of the overall system, as well as the individual components so that they can be efficiently activated and deactivated.