Accelerating AAL Home Services using Embedded Hardware Components

Georgios Keramidas, Christos Antonopoulos, Nikolaos S. Voros
Embedded System Design and Application Lab
Computer and Informatics Engineering Department
Technological Educational Institution of Western Greece, Greece
{gkeramidas;cantonopoulos;voros}@teiwest.gr

Fynn Schwiegelshohn, Philipp Wehner, Michael Huebner
Ruhr-University, Bochum, Germany
{fynn.schwiegelshohn;philipp.wehner;michael.huebner}@rub.de

Diana Göhringer
Technische Universitaet Dresden, Germany
diana.goehringer@tu-dresden.de

Evaggelinos Mariatos
AVN Technologies, Cyprus
emariatos@avntechgroup.com

Corresponding author's email: gkeramidas@teiwest.gr

Abstract. The EU-funded project RADIO brings forward a new health care paradigm according to which a mobile robotic platform can act as assistant to an elderly person in his/her domestic environment. The main goal of the robot is to detect ADL (Activities of Daily Life) related to basic self-care tasks, such as sleeping and taking medications, as well as instrumental ADL related to housework. ADL detection is based on visual, depth, and audio signal analysis as well as their fusion. However, robot assistance in everyday living suffers from limited autonomy dictated by the robot battery e.g., the robot has to constantly know where the person is and to be able to move if the person moves to a new location or another room. In this chapter we present the line of research followed in

the RADIO project in order to reduce the usage of the power-hungry processing components and, consequently, the need for revisiting, to the extent possible, the robot charging station. Our approach is based on building specialized, hardware acceleration units on a Zynq-based FPGA of Xilinx. Moreover, a hardware-software partitioning approach is performed base on the HLS (high-level-synthesis) paradigm. In this way, the robot will be able to perform computation intensive tasks in a power efficient manner.

1. Introduction

RADIO aims to provide a real-life solution to support elderly people in their domestic environment **Error! Reference source not found.**. The core concept and approach of the project has been presented in more details in [2]. It is important to note that the RADIO project emphasizes on being unobtrusive and well accepted while remaining fit for its clinical purpose. Technically, these requirements pertain to user interfacing, specifically adapted to the elderly; ethically and clinically adequate data collection, transmission and processing; integrated and power aware data collection, transmission and processing; and an efficient and flexible architecture that can integrate heterogeneous health and comfort-related devices.

In the context of RADIO, the robot acts mainly as a mobile sensor platform that monitors and guides an elderly or disable person throughout the entire day, using cameras and microphones as well as through direct access to the home automation infrastructure.

However, to achieve its mandate, the robot has to constantly know where the person is and to be able to move if the person moves to a new location or another room. While monitoring, the robot mechanical systems do not operate, thus saving battery life. But this is not enough; the power consumed by the main robot controller is significantly high, given that (even when not moving) a relatively powerful computing engine needs to be active. This holds true even during idle time periods e.g., when the person is sleeping or watching TV, as there is no way for the robot to know if and when the person intends to move. To reduce this

idle-time power consumption wastage, as part of the RADIO project, a hardware (HW), FPGA-based implementation of event detection algorithms is set forward.

2. Data Collection and Processing

Assistive environments are typically implemented as smart homes or similar sensor networks that collect and analyze data related to mood and activities of daily life (ADL), as well as automatically providing notifications to care-givers when specific events are identified such as falls and similar emergencies. This is a well-studied subject with a rich literature and developed systems. One major line of research takes advantage of wearable sensors or sensors embedded in household items and appliances in order detect a wide range of ADLs [11][12][13][14]. Such approaches are not well aligned with the objective of developing a system that is unobtrusive by, among other features, also avoiding requirements on what the end-users should wear or use.

RADIO concept follows the line of research that uses computer vision and audio analysis to recognize interesting events. In fact, one of the main outcomes of the project is our analysis of the extent to which unobtrusive monitoring is adequate to meet strigent clinical requirements. To the best of our knowledge, the presented methodology is unique in assisted living environments; we do not assume as a goal the maximal detail and accuracy that state-of-the-art sensing hardware can achieve. Instead we assume as a goal, the collection of the minimum amount of sensitive content and personal information, at the minimum obtrusion, that will allow medical personnel to make informed decisions.

Naturally, such a broadly defined goal needs to be refined to reflect the societal impact that RADIO aims to achieve: allowing elderly people with mild cognitive impairment to maintain an independent life, at their own home, for longer than what is safely possible today. In order to have a guideline about what information is used by medical doctors to assess such conditions, the interRAI Long-Term Care Facilities Assessment System (interRAI LTCF) has been thoroughly analyzed. interRAI LTCF

enables comprehensive, standardized evaluation of the needs, strengths, and preferences of persons receiving short-term post-acute care in skilled nursing facilities as well as persons living in chronic care and nursing home institutional settings. As a result, we have determined the mood and ADL recognition items that can be immediately and automatically extracted and those that appear to be outside the reach of the current state-of-the art.

Some characteristic examples include clothes change detection, where recognizing the end-result of having changed clothes [5] is sufficient while the detailed capture of all the motions performed in order to change clothes offers no medically relevant information. Similarly, depth camera data can be used to detect potentially dangerous activities [6] and food intake [7]. Moreover, acoustic processing can provide information about several ADLs, such as hygiene, washing and walking [8]. Finally, mood can be inferred from both visual and audio analysis [8][9][10].

These observations have led us to a design where the main data collection device is a mobile robot equipped with audio, visual, and depth camera. The mobility of the robotic platform is important for placing the sensors at positions that offer the maximum level of detail (e.g., full-face images for recognizing mood from facial expressions) without placing requirements of the end-user such as having to move in front of a sensor or having to use specific equipment. This choice, however, also introduces challenges due to the heterogeneity of the home automation and robotics infrastructures, as well as due to the low power consumption requirements necessary for having a mobile platform with sufficient battery autonomy.

3. The AAL Robot as a Heterogeneous Processing System

The RADIO robot is outfitted with two processing units: an Intel NUC which is responsible for controlling sensors and actuators and an Avnet PicoZed equipped with Xilinx Zynq-7000 all programmable System on Chip

(APSoC) [4]. Additional devices include an Asus Xtion Pro camera and a Hokuyo laser scanner [3]. 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

External interfacing is achieved through RF networks with emphasis on low power and minimal usage of the RF spectrum. Adopted communication technologies are not suitable for conveying real time streaming data like audio or video, leaving no other option, but to perform the processing of the multimedia workloads in the robot.

As mentioned, our AAL approach heavily relies on the acquisition and processing of audio and video streams for analyzing and recognizing activities of daily life (ADL) [5][6][7]. Recognizing the emotional status of patients and the identification of emergency situations, such as the detection of falls, are also of high importance [8][9][10]. As a result, the main processing engine(s) of robot are designed to operate as a power-efficient architecture for streaming data processing.

Apart from the FPGA fabric, the Xilinx PicoZed platform includes also an ARM Cortex A9 dual core processor (equipped with a Neon co-processor) interfaced with programmable logic.

4. Fast and Energy Efficient Data Processing

Figure 7.1 illustrates two different scenarios of positioning the robot processing elements and their interfaces. In both cases, the bulk of processing is indented to be performed on the FPGA platform. However, the two scenarios differ in the points where the sources of audio

(microphone) and visual (camera) streams are located. The advantages and disadvantages of each scenario are analysed in the rest of this section. In any case, the camera and audio data streams will be continuously monitored 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 NUC, in the Zynq ARM processor, or accelerated with fixed logic or reconfigurable hardware components inserted in the FPGA reprogrammable logic.

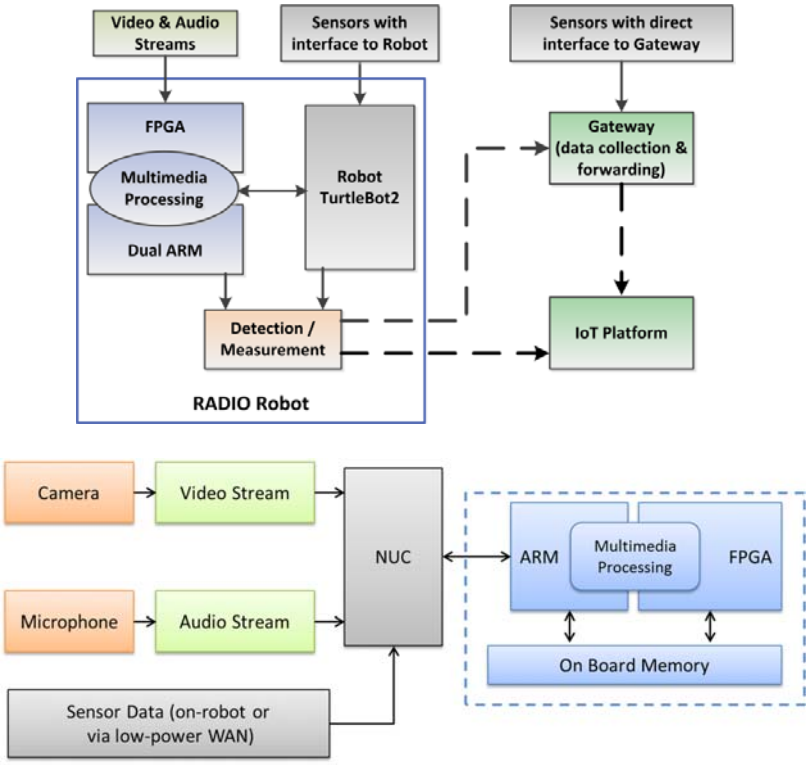

Figure 7.1: Diagram illustrating two different scenarios of the data flow between the RADIO robot, gateway, and IoT platform

In the second scenario depicted in Figure 7.1, the camera and audio data streams, as provided through the NUC, will be continuously monitored by the processing elements of the FPGA platform. Therefore, a pipe that brings the streams down on the FPGA platform is implemented on the NUC using ROS channels. To avoid overloading the interface with unnecessary memory transfers, the ROS channels are constructed so that image and audio data is provided to the FPGA platform on a need-to-know basis. When the captured image data are transferred to the FPGA device (through ROS messages), the RAM which resides on board will be directly accessible from both the dual ARM core and the FPGA fabric (to minimize on-board memory transfers).

Figure 7.1 illustrates the streaming data flow through the robot processing elements and their interfaces. The camera and audio data streams are continuously monitored and when activity is detected the corresponding algorithms (which can analyze and recognize the activity) are triggered.

Depending on the specific combination of algorithms that get triggered, some or all computational tasks may be executed in the i) NUC, ii) Zynq ARM processor, or iii) accelerated by hardware components in the FPGA.

To provide a specific example of this concept, we describe here the implementation for the algorithm used in ADL “Measure time to get out of bed”. Data processing with this algorithm consists of following (simplified) steps:

- For each Frame
    - Store frame in RAM to be used as “previous” in next iteration
    - Split Frame in blocks of 9x9 pixels
    - Compare blocks with “previous frame” blocks to detect difference
    - Get bounding rectangle of all different blocks
    - Get centre of all different blocks
    - Check bounding rectangle and centre against the set limits
    - Report any events / findings

To optimise these steps, we modify the execution order and assign to various blocks on the above architecture as follows:

```
• For each frame (row 1 to row 469)
    o Get next 9 rows [Camera -> Video Stream -> NUC -> ARM]
    o For each row:
        ▪ Split in 9x9 blocks [ARM->OnBoardRAM]
        ▪ For each block:
            • Read 9x9 [OnBoardRAM->FPGA]
            • Compare and store “previous” [FPGA]
            • Report per-block result [FPGA->ARM]
    o Calculate Bounding and Centre [ARM]
    o Report to NUC [ARM->NUC]
```

This simplified example illustrates how a method can be adapted to fit the specific requirements of the architecture, achieving optimized memory transfer, speed, and power consumption.

The potential benefits inherently offered by the first scenario (illustrated in Figure 7.1) are further analysed below.

4.1. Concept within the distributed RADIO environment

The need for fast and efficient processing of the data streams in the distributed RADIO environment comes from the need to be:

- *Responsive*: in some cases, the data collected on the robot and the measured/detected quantities and events are correlated centrally with data from the smart home sensors or other sources. For example, detection of exiting the room can be correlated with a motion sensor mounted on the room walls
- *Energy Efficient*: as the robot relies on battery, data processing should be limited to what is necessary and executed on the node yielding the lowest overall power consumption

To put this into perspective, an example state-diagram of the robot is presented in Figure 7.2.

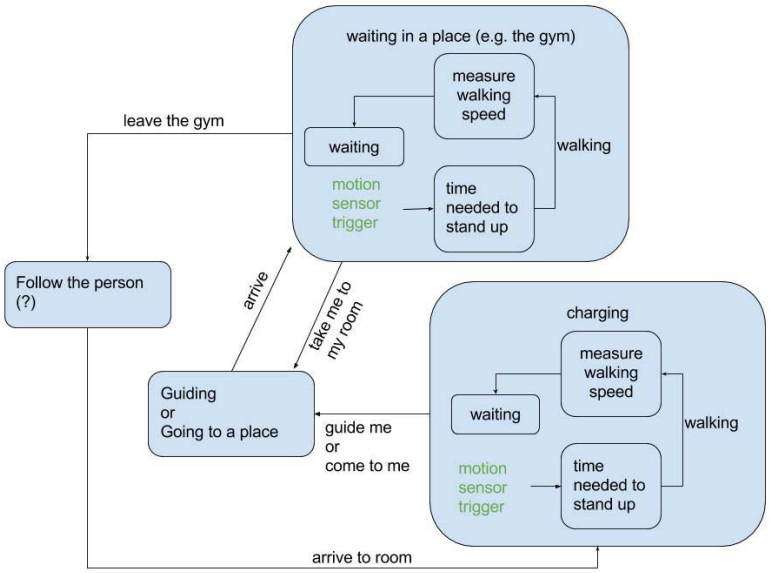

Figure 7.2: State diagram highlighting different (power) modes

In this diagram, we have two points (in green fonts) where entering is triggered by the smart-home infrastructure through getting the event from a motion sensor. There are also a number of states that involve heavy data processing and we have to ensure that this is either:

- very low power
    - time needed to stand up (gym)
    - measure walking speed (gym)
    - follow the person
    - guiding or going to a place

  or
- executed when the robot is stationed on its charging station
    - time needed to stand up (in the room)
    - measure walking speed (in the room)

4.2. Modeling the operational states of the AAL robot

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)

Table 1.1: Robot Subsystem Energy Usage

State	CPU	FPGA	Sensor	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/ Changing	Used	Used	Used	Not used	Used

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 7.1 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 2.2: Energy profile by using HW accelerator

State	CPU	FPGA	Sensor	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/ Changing	Used	Used	Used	Not used	Used

A more detailed description of the cases is illustrated below:

- *Waiting State*: The FPGA can connect only to an 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.
- *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 7.2.

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

In the context of RADIO, the dedicated HW components (see Section 6) are implemented in the programmable logic (PL) of the Picozed APSoC using the Vivado HLS (High-level Synthesis) tool. The accelerators are optimized for performance and area and they proved flexible enough to perform their role: *early detection of a high-possibility for an event so that SW-based processing can be invoked*.

5. Experimental Profiling and Results

5.1. Example ADL use cases

To prototype and experiment with the alternative approach discussed in this chapter, we selected a small number of ADLs as target use cases for the monitoring state of the robot. The selected ADL recognition methods must:

- Be able to be detected when the robot is not moving
- Have a significant part of pre-processing, which can be done on the FPGA
- Get triggered by some external signal or some very simple HW-only method
- Not rely on information exchanged over the WiFi or Smart-home Network

The selected methods will be the ones which detect:

- *The time that is 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 for the acceleration within the FPGA hardware. This algorithm divides the image into different regions. If the center 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 complex algorithm.

The 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, for the above mentioned ADLs:

- For the time-to-stand-up ADL, the hardware component will collects and calculates data from all regions, providing a trigger to software 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; a movement that might create false positives. An always running HW component will be monitoring such small movements and constantly re-center the view.

5.2. Optimizing for power

In order to determine the SW-HW co-design of the FPGA-ARM system, extensive profiling of the image processing algorithms is needed (see Section 6). Generally, FPGAs provide high-performance when manipulating the images pixel-wise or in small blocks. This allows several hardware implementations with different degrees of parallelism. Each hardware design then must be evaluated within the overall RADIO framework in order to determine the best implementation.

At the first phase of the project, our work was allocated to have a working HW acceleration framework, and not to the specific optimization of each HW component. In a subsequent phase, we profiled three options, so that the expected benefits of various optimization approaches can be

quantified, allowing to focus on these solutions that will yield the most benefits.

The three analyzed options – as described earlier– are:

- *No offloading*, 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 power down)

5.3. Profiling on daily activity use cases

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:

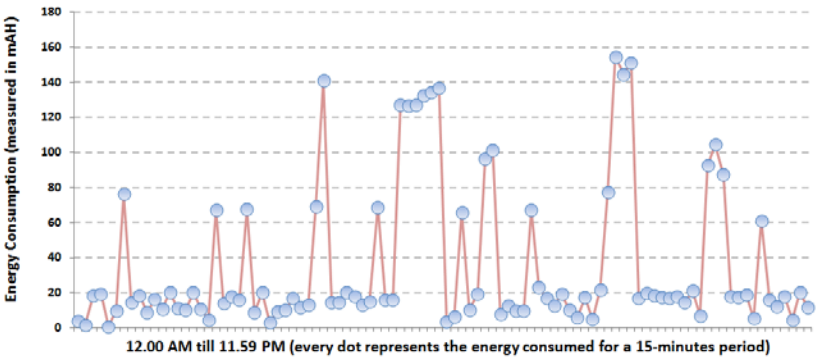

Figure 7.3: Daily activity profile

- *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 7.3. More specifically, the data presented in Figure 7.3 are 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.

5.4. Power-savings results

The three options analyzed in the previous section are then tested on the profile illustrated in Figure 7.3. 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 7.3 (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 7.4. The vertical axis in Figure 7.4 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 platform (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.

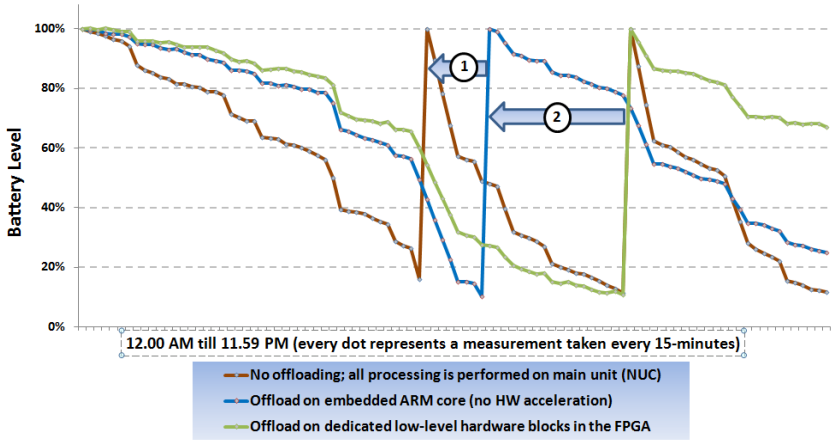

Figure 7.4: Run-time depletion of robot battery and number of required charges for the three studied offloading policies

As Figure 7.4 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 7.3). 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.

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 7.5: 7.5 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. 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 of 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 of 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.

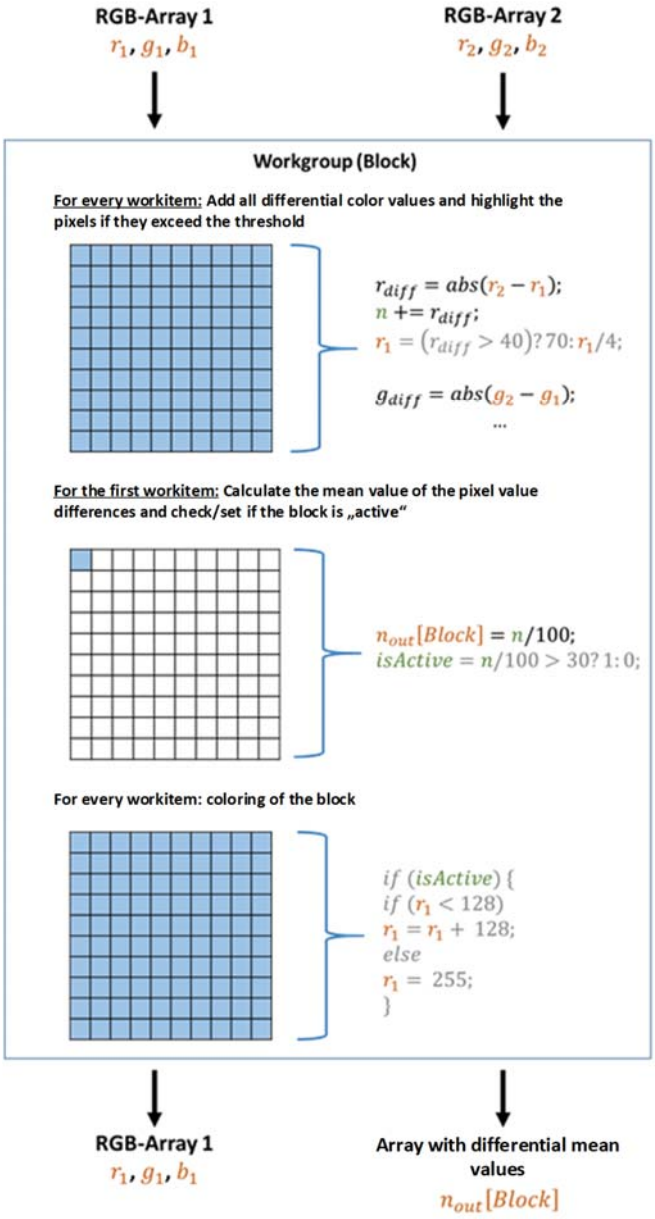

Figure 7.5: Schematic view of the kernel design annotated with pseudocode

6.1. 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 a 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 3.3: Subblock of the algorithm

7.3.

Table 3.3: Subblock 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 7. for the algorithm with markups and in Figure 7.7: 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 7.6: 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 7.7: 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.

6.2. 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 values 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 7.8: shows the kernel implementation as schematic and as pseudocode.

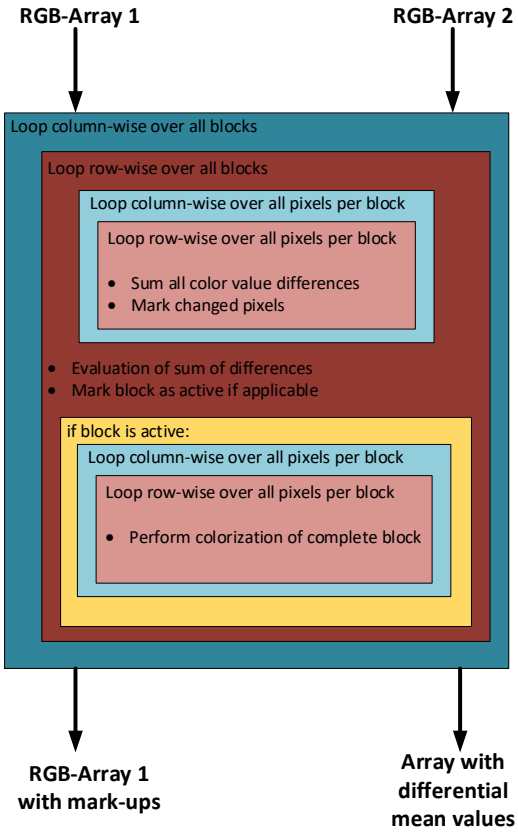

Figure 7.8: Depiction of the implemented algorithm

The initial version of the OpenCL code can be generated with 100 MHz. Figure 7.9 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 or (at 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 7.9: Resource utilization 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 burst mode 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 7.10:. Here, the number of used BRAM blocks has increased from 2 to 74 while all other resources 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 7.10: 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 7.11:. 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 a 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 7.11: Resource utilization of the OpenCL kernel after optimizing 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 7.8:. 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 7.12: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 7.12: 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.

6.3. 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 4.4: Energy profile by using HW accelerator

7.4 shows the execution times of the different 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 4.4: Energy profile by using HW accelerator

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. Summary

This chapter presents the system-level approach followed in the RADIO project targeting to increase the autonomy (measured in terms of battery charges) of the robotic platform in AAL environments. The proposed approach is benchmarked in experimental conditions via use case profiling and it is driven by the daily activity patterns of the target elderly or disable people. The daily activity patterns were collected by care-givers personnel during the third pilot phase of the RADIO project. As part of this, a general behaviour concept is presented in this book chapter, which allows the robot to switch between extremely low power states and active state when necessary based on user behaviour.

Special emphasis is given to fully utilize the robot processing units in the most efficient manner (APSoC with an ARM-based processing system and a FPGA programmable logic). This enables the robot to perform computation intensive tasks very efficiently and very fast. However, in order to extract the most performance out of these processing units, all tasks that need to be executed by the robot platform have to be analysed and scheduled accordingly on the processing system or on the programmable logic.

Finally, the FPGA implementation of an image processing algorithm for monitoring the state of the patient is also presented. All aspects of the implementation are provided, while detailed statistics (FPGA resource utilization and speedup) related to the efficiency of the FPGA acceleration are also offered. The design of the accelerator followed the hardware-software partitioning approach and it is based on the HLS (high-level-synthesis) paradigm in order to reduce the programmability and development effort.

References

[1] EU project RADIO. http://www.radio-project.eu

[2] C. P. Antonopoulos, G. Keramidas et al. Robots in Assisted Living Environments as an Unobtrusive, Efficient, Reliable and Modular Solution for Independent Ageing: The RADIO Perspective. Proc. of Intl. Sympocium in Applied Reconfigurable Computing, 2015.

[3] TurtleBot2 Robot Development Kit. http://www.turtlebot.com

[4] Xilinx Inc. Zynq-7000 All Programmable SoC Overview, 2015.

[5] D. Sgouropoulos, T. Giannakopoulos et al. Clothes change detection using the Kinect sensor. Proc. of Intl. Conference on Signal Processing and Multimedia Applications, 2014.

[6] C. Zhang, Y. Tian. RGB-D Camera-based Daily Living Activity Recognition. Journal of Computer Vision and Image Processing, 2012.

[7] H.M. Hondori, M. Khademi et al. Monitoring intake gestures using sensor fusion (microsoft kinect and inertial sensors) for smart home telerehab setting. Proc. of Annual Healthcare Innovation Conference, 2012.

[8] M. Vacher, A. Fleury et al. Complete sound and speech recognition system for health smart homes: application to the recognition of activities of daily living. New Developments in Biomedical Engineering, 2010.

[9] S. Koolagudi, K.S. Rao. Emotion recognition from speech: a review. Journal of Speech Technology, 2012.

[10] Z. Zeng, M. Pantic et al. A survey of affect recognition methods: Audio, visual, and spontaneous expressions. Transactions on Pattern Analysis and Machine Intelligence, 2009.

[11] A. Matic, P. Mehta et al. Monitoring dressing activity failures through RFID and video. Methods of Information in Medicine, 2012.

[12] M. Philipose, K.P. Fishkin et al. Inferring activities from interactions with objects. Pervasive Computing, 2004.

[13] D.J. Cook, M. Schmitter-Edgecombe. Assessing the quality of activities in a smart environment. Methods of Information in Medicine, 2009.

[14] A.F. Dalton, F. Morgan, G. Olaighin. A preliminary study of using wireless kinematic sensors to identify basic Activities of Daily Living. Proc. of Intl. Conference on Engineering in Medicine and Biology Society, 2008.