Research Article

Design of a flexible reconfigurable mobile robot localization system using FPGA technology

Agnès Ghorbel[1] · Nader Ben Amor[1] · Mohamed Jallouli[1]

Received: 18 December 2019 / Accepted: 27 May 2020 / Published online: 7 June 2020

Abstract
Mobile robot activities have variable computational requirements according to various parameters related to the robotic applications behaviour and the robot dynamic environment. This high variability puts many challenges for a complete robot navigation system design. With their high flexibility, reconfigurable architectures are a suitable choice to design such dynamic system. Mobile robot navigation steps (perception, localization, obstacle avoidance and path planning) are extremely complex to be embedded in a complete navigation controller. In this paper, we presented a generic and a flexible platform based on FPGA technology, for studying and prototyping complete unified robot navigation system. We exploited the dynamic reconfiguration mechanism that allows modifying the navigation controller according to different navigation algorithm requirements. We showed the effectiveness of this platform in the localization step. We have a dynamic reconfigurable system that switches between three different localization techniques to satisfy energy/surface ratio and exploit the released resources to implement other robot tasks.

Keywords Robotics · FPGA · Dynamic reconfiguration · Hybrid localization

1 Introduction

1.1 Context

Navigation in mobile robotics is a methodology allowing the guidance of a Mobile Robot (MR) to perform, well and securely, a task through an environment with obstacles. The success of this task requires an adequate coordination between the four major blocks involved in navigation: perception, localization, path planning and obstacles avoidance. The MR should possess an architecture able to coordinate the on board navigation elements in order to correctly achieve the different objectives specified in the mission with efficiency whether in indoor or outdoor environments. One efficient way of building a robotic platform is to join pre-built robot parts, for example acquire a MR with desired locomotion, pre-built sensors with specific interface, robotics arms, cameras, etc. In this way, we can build a complex custom robot. A complete navigation system is still required. It has to be modular, scalable and flexible. The different navigation steps have high computational requirements. They require hardware oriented architectures and flexible software oriented architectures, at the same time. Reconfigurable architectures are therefore an adequate solution. FPGAs are a good choice, because they can implement a processor-based system or a microcontroller-based system or a combination of both. Moreover they can implement complete embedded systems on a chip, co-processors, custom user logic, communication protocols or multi-processor systems.

✉ Agnès Ghorbel, ghorbel.agnes@gmail.com | [1]Computer and Embedded System Laboratory, Ecole Nationale d'Ingénieurs de Sfax, Université de Sfax, Sfax, Tunisia.

1.2 Background

These days, intelligent mobile robots are expected to perform more and more complex tasks in different application areas. An enormous amount of research and many studies have been conducted to focus on the navigation problem, obstacle avoidance, wall following, and intelligent parking. To attain the desired objectives, mobile robots use different sensors types to collect environmental information and actuators set for their movement and reaction. Accordingly, the choice of the electronic components must be rigorous to accurately control the wanted position and route, perform real time data processing, and do multiple tasks.

The standard software based control platforms like Digital Signal Processors (DSP), still the popular choice to carry out robot control operations [1, 2]. DSP is a dedicated microprocessor generally programmed in C language to enhance performance. The major drawbacks of such these architectures are: (a) puissant DSPs are expensive and their related software applications may not reflect the hardware performance; (b) sequential task processing; (c) trouble to make changes in the hardware; and (d) lack of design portability [3]. In addition to that, the used techniques and the platforms are limited since they don't provide other services. Thus, it is better to aim a SoC architecture with CPU and custom coprocessors that offers the ability to execute more applications. Recently, wireless sensors are used in mobile robot navigation to improve the effectiveness and the robustness of robot localization [4].

The FPGA is getting particular attention from the scientific community to address computational constraints. This implies that intensive computation should be applied to the computer vision, image processing, robotics systems, etc. FPGAs offer high performance close to ASICs. However, unlike ASICs, FPGAs provide a high degree of flexibility similar to a general purpose computer. FPGA-based systems are faster than a pure software approach in terms of computational power [5]. Thus, an infrastructure allowing for the exploitation of the high performance capability of a hardware implementation with the benefits of an autonomous resource management is a promising alternative to be used. Many robotic applications are implemented on FPGAs. Most of these FPGA implementations focus on a simple task of the whole robotic system or use the FPGA for a specific and particular application. In [6], an embedded fuzzy controller on FPGA devices was described. The design methodology and tool chain presented by the authors were applied to achieve a control system for solving the navigation tasks of an autonomous vehicle. The authors in [7] describe a hardware architecture for implementing a sequential approach of the Extended Kalman Filter which is suitable for mobile robotic tasks. In [8], the authors proved the effectiveness of using the FPGA as a coprocessor together with the dual core processor to speed up neural calculations. In [9], the proposed robot navigation is developed with modified approach of depth first search algorithm for patrolling services. It is implemented using FPGAs for successful navigation. An FPGA will works as control unit, after sensing free path it drives the robot as per algorithm. The authors in [10] present the design and testing of a newly developed field programmable gate array (FPGA) board for mobile robot research. The idea consists of interfacing the FPGA to a 16-bit digital signal controller (dsPIC). It's used for exploiting position tracking of the used mobile robot. The goal of authors in [11] is to bring MATLAB and FPGA on to the same platform to develop image processing algorithms for edge detection in order to actuate the motion control of the robot in the environment. As concluded by authors in [12], FPGA is an ideal choice for implementation of visual navigation for real time moving object tracking algorithms. In the study proposed in [13], the design and control of an FPGA based four-wheel drive mobile robot that detecting obstacles and avoiding them were carried out. It is seen that the mobile robot, which is controlled by FPGA, quickly detects the obstacles and changes its direction according to this obstacle. In [14], the authors present a novel use of open-source FPGAs for educational robotics using a new visual language for robot programming. The objective of authors in the study [15] was to develop a hybrid CS-fuzzy optimization technique for application to evolutionary real-time fuzzy control omni-Mecanum-wheeled autonomous vehicles. With the advantages of FPGA, metaheuristic algorithm, and fuzzy theory, the proposed CS-fuzzy control method outperforms the traditional fuzzy controllers. Many FPGA-based solutions have been implemented in the field of robotics but in our best knowledge, there is no prior work on FPGAs for complete robotic navigation systems that includes the afore mentioned four major conventional navigation blocks.

The design and implementation of hardware efficient solutions has been previously investigated for specific robotic tasks. Thus, our work proposes a generic and flexible platform based on the FPGA technology, to study and design a complete robot navigation system with different computational requirements. This platform can be used to address the largely different computational requirements of robot navigation applications. In fact, energy efficiency was ensured through using an FPGA low complexity with limited logic cells. Furthermore we resorted to the use of the dynamic reconfiguration to embed all complex mobile navigation activities on the robot. A little research has been carried out in the area of partial reconfiguration for mobile robots. To achieve an intelligent, task-based reconfiguration, the authors in [16] introduced a novel methodology for the computational hardware for MR applications using FPGAs. Hardware/software co-design and hardware reconfiguration were used to design fault-tolerant and reliable robotic systems. The authors in [17] propose an implementation for a highly customizable FPGA-based vision processing module for mobile applications. The partial dynamic reconfiguration is used in image processing tasks. If a specific processing task is finished, a new hardware configuration can be loaded to optimize the resource utilization on the fly. In [18], the authors focus on the dynamic reconfiguration of the robot's FPGA hardware to locally perform image and video processing tasks. The main aim of the work is to create a set of reconfigurable hardware configurations for a color recognition module supporting different application parameters. In [19], the authors implemented a line follower robot for a white line and a black one. The robot has to dynamically reconfigure the FPGA during run-time while the robot senses black line after white line or vice versa. However, there are no prior works on FPGA reconfiguration in a robot localization system. So, our approach is to propose a global modular FPGA based system for robot navigation where the FPGA is the robot'brain' to manage complex tasks that other embedded platforms cannot guarantee especially for complex vision applications. We also exploit the dynamic reconfiguration feature that allows modifying the navigation controller according to the requirements of the different navigation algorithms. A dynamic reconfiguration can be used to enhance the FPGA resource utilization by time-sharing the reconfigurable resources among different designs and applications needs.

The remainder of this paper is divided into five sections. Section 2 presents the complete robotic platform and its components. In Sect. 3, we describe the localization techniques used in robotics and detail each technique with its appropriate design architecture. Section 4 presents the remote control system and how it is achieved. Finally, Sect. 5 displays the main conclusions of the paper.

2 Embedded robot navigation platform overview

Our MR navigation includes different interrelated activities: perception, localization, obstacle detection and remote tracking/control: (i) Perception consists in obtaining and interpreting sensory information; (ii) localization provides the strategy to estimate the robot position within the spatial map; (iii) obstacle avoidance deals with the strategy of detecting and avoiding obstacles (v) path planning allows the control of the MR at real time. Figure 1 describes this complete platform.

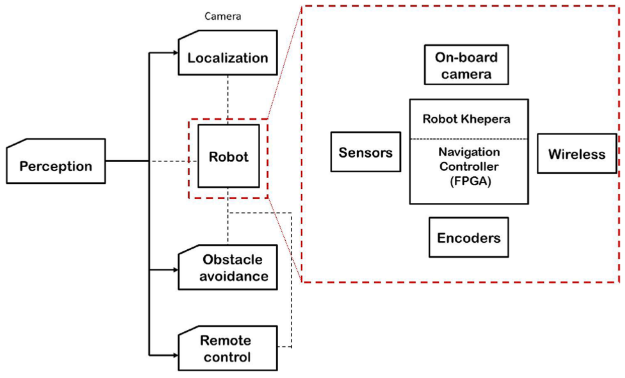


Fig. 1 The proposed platform for robot navigation system

The robot consists of:

- The MR Mini Khepera-II [20].
- The ML507 as the navigation controller: it integrates an embedded PowerPC hard-core CPU cameras: One is used to track the robot and participate in its localization and the other is mounted onto the robot itself to detect obstacles during navigation with a clock frequency up to 700 MHz, a Micro-Blaze soft-core CPU with a clock frequency up to 125 MHz, different RAM memories (256 MB DDR2, 1 MB SRAM), VGA input, DVI output, RS-232, JTAG interface, Ethernet port and more than 11,000 slices of reconfigurable logic.
- On-board sensors: odometers to measure the robot relative pose and infra-red leds to detect obstacles.
- Surrounding cameras: One is used to track the robot and participate in its localization and the other is mounted onto the robot itself to detect obstacles during navigation.

The Linux operating system (OS) was chosen to manage the navigation controller. It would lead to increase prototyping convenience. Using Linux, an open, robust and well documented OS, can provide a rich set of debugging and additional support software like drivers and APIs, especially for network functionality. Network communication management is so complicated without using the Ethernet interface provided by the Embedded Linux. It also provides rich possibilities to include libraries (like image processing library: OpenCV) that can be cross compiled under the PPC.

In this paper, we first showed the prototype results of the navigation platform. The supported functionalities of this prototype are: robot localization using ceiling camera, distant web access and the support of dynamic reconfiguration.

3 Localization concept

We exploit the dynamic reconfiguration benefit for the localization step. In our robot, three localization methods were applied: the relative, the absolute and the hybrid. Table 1 presents a summary on localization techniques.

According to this table, the stand-alone sensor cannot exceed certain physical limitations such as limited range and field of vision. Consequently, combining information from different sensors would expand the area around the vehicle and improve the reliability of the entire localization/navigation system in case of a sensor failure.

The proposed approach developed in this paper used two different sensors. One is based on encoders to insure relative pose using the robot kinematic model and fuzzy controller to reach a target. The other is based on a camera placed on the ceiling of the robot navigation environment to provide the absolute pose. The advantage of using such an absolute system is to automatically generate the robot initial coordinates and reduce the errors in encoders' measures through retiming the visual control. We can also note that the relative localization is quite simple while the absolute one requires high computational power. Thus, the proposed concept is to integrate the advantages of a localization based on an odometer and a localization based on camera data and make them complementary which will enable the robot to accurately localize itself. Both calculations are carried out on the FPGA as the robot microcontroller does not deal with these tasks. Regarding the camera, it's linked to the FPGA via the VGA input in the actual prototype. This solution is suggested owing to the available ports of the ML507. A wireless camera is the best alternative and there is no need for a high bandwidth as a slow frame rate is enough to correct the robot position. Each localization technique has both advantages and drawbacks and should be applied in a specific circumstance. To allow a flexible robot localization/navigation, the different techniques should be permanently alert for use.

3.1 Relative localization

The experiments were performed on the MR with two independent driving wheels which movements were controlled by varying the speed of each wheel. The position is obtained according to the robot internal mathematical kinematic model. The treatment is carried out using a first architecture noted as "archi loc rel" made up of the PPC processor, one BRAM (a 128 Kb internal FPGA memory), and an UART controller to ensure serial communication with the robot. Figure 2 illustrates the "archi_loc_rel".

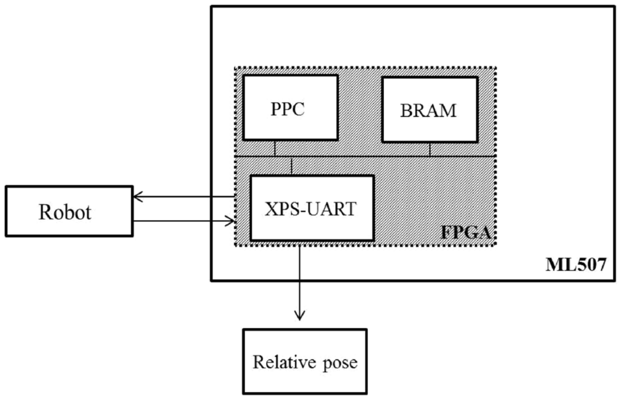


Fig. 2 The design architecture for relative localization

Table 1 Summary of localization techniques

Localization techniques	Advantages	Disadvantages
Relative Localization (Odometers, Gyroscopes,…)	Easy to accomplish Simple Inexpensive [21]	Imprecision in measures Accumulating errors [22] Providing information only about the internal state of system
Absolute Localization (Infrared sensors, camera, laser, etc.)	Independence in position estimates [23] Minimizing error More accurate view of the robot in its environment External robot information	Dependence on the environment Complex Very slow Greater likelihood of failure
Hybrid localization (data fusion)	Reducing uncertainty [24] Increasing the dimensionality of the measures Improving resolution [24] Reduction in time	Much calculation to perform a task Short range of sensors Filter divergence Necessity to know initial pose

Table 2 Synthesis results of "archi_loc_rel"

Device		Virtex-5 xc5vfx70t	
Resource	Total	Usage	Usage percent
Device utilization summary			
Slice Registers	44.800	522	1
Slice LUTs	44.800	452	1
Number used as Logic	44.800	429	1
Number of occupied Slices	11.200	390	3
Total Memory used (KB)	5.328	1.152	21
Timing summary			
Minimum period			2,5 ns
Maximum frequency			400 MHz

The FPGA resources occupation of "archi_loc_rel" is presented in Table 2. Less than 3% of the FPGA resources are required. The measures based on the odometer modules are erroneous, mainly when the robot must travel distances. This is due to the wheels' slippage and drift problems. To address this problem, we opted to correct these measures by external data coming from a camera placed in the robot parallel plan.

3.2 Absolute localization

The absolute position is performed on the robot. It is achieved by applying image processing algorithms to the robot frames coming from a ceiling camera. The camera acquisition and display module are managed by the Microblaze processor while the absolute localization algorithms are executed by the embedded PPC processor.

3.2.1 The camera acquisition module "cam"

The CAM module includes:

- The VGA input of the ML507 that must be connected to a VGA source.
- The DVI output of the ML507 that must be connected to a DVI/VGA compatible screen.

A ceiling camera with a resolution of 640 * 480 has been used. Its S-video output is connected to the VGA input of the ML507 using a video converter. The CAM architecture is as shown in Fig. 3.

- The microblaze processor
- A MicroBlaze Debug Module (MDM).
- The VGA input IP block: this block consists of AD9980 analog interface that converts the incoming signal from the ML507 VGA input into a YUV signal and of a VFBC (Video Frame Buffer Controller) which allows the user to read and write data.
- The XPS TFT IP block that sends the output signal to DVI output connector of the ML507.
- A Multi-Port Memory Controller (MPMC) interface.

Table 3 summarizes the used capacity in the FPGA for the CAM using Xilinx Platform Studio (XPS) tool.

From Table 3, we remark that about 45% of FPGA slices were occupied. We have to manage the existent resources between different designs needed by different applications.

Table 3 Synthesis results of camera acquisition module

Device		Virtex-5 xc5vfx70t	
Resource	Total	Usage	Usage percent
Device utilization summary			
Slice Registers	44.800	9.301	20
Slice LUTs	44.800	8.658	19
Number used as Logic	44.800	7.930	17
Number of occupied Slices	11.200	4.803	42
Total Memory used (KB)	5.328	1.782	33
Timing summary			
Minimum period			8 ns
Maximum frequency			125 MHz

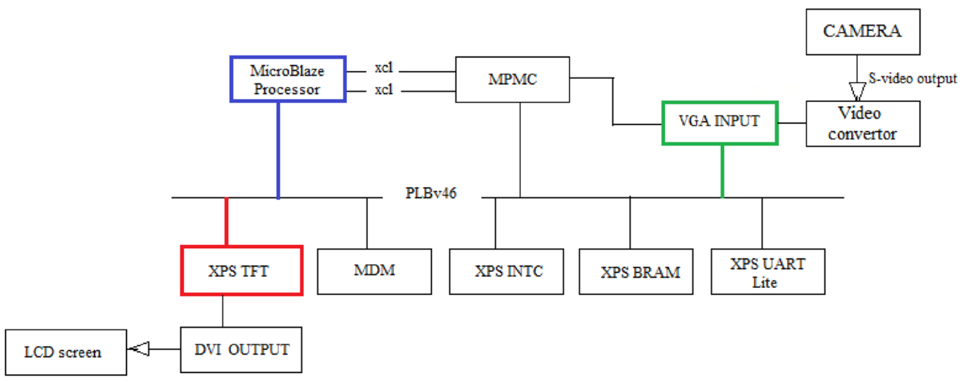


Fig. 3 The camera acquisition architecture

3.2.2 The robot localization module "RLM"

To ensure a continuous robot navigation, we would rather to execute the RLM on a PPC processor as it is more performant than the MicroBlaze. The RLM is managed by the Embedded Linux OS. It consists of a pipeline of image processing steps. To facilitate the RLM implementation, the OpenCV library was used. It was cross-compiled with Linux for PPC.

The RLM processed the input robot images via a straightforward pipeline to determine the location of the circular shaped robot. We give here a summary of the technique. The whole technique is presented in [25]. Firstly, the Cartesian coordinate system has to be set up using the four landmarks. Secondly, the robot has to be detected and finally its coordinates are computed. The robot detection consists in the identification of its circular shape in the input frame. To reduce the RLM complexity, we used a window embracing the robot (like the search window in video compression application) and we searched for the circular shaped robot instead of scanning the entire input image. The window size is fixed to 70*70 pixels after several tests on various robot speeds and motion orientations. The 70*70 is the minimal window size to cover the robot area and the maximal to treat the minimum pixels number.

The pipeline as shown in Fig. 4 consists of a grayscale conversion (step 2). A Sobel filter (step 3), whose output was further improved by a specified threshold value equal to 63 (step 4). This value is calculated using an iterative method. Afterwards, the image was smoothed by an erode filter (step 5) and finally a Hough Transform (step 6) was used to extract the circle corresponding to the robot.

The RLM execution time on the PPC (400 MHz, fixed point) is 2.187 s. This time is not sufficient to ensure real time navigation which has to be less than 0.48 s [26].

3.2.3 Reducing RLM processing time

Three techniques are used. The first technique is to raise the PPC frequency from 400 MHz to 600 MHz (the maximum value authorized by ML507) using a PLL module. We also add a floating point co-processor to the PPC as many operations of the RLM are in a floating point format. The second technique is based on the use of a hardware accelerator to execute the computationally complex task of the RLM in hardware (using FPGA logic blocks). To select which RLM task is a candidate for a hardware implementation, we profile the RLM algorithm. Figure 5 shows the time consumption rates.

Through profiling results, the edge detection function consumes 60% of the total time since it comprises complex operations like Eq. (1).

$$|\mathrm{Mag}| = \sqrt{E_h^2 + E_v^2} \tag{1}$$

where Mag: The gradient magnitude, E_h the intensity along horizontal directions, E_v the intensity along vertical directions.

This function is implemented as a hardware accelerator named "Acc-Block". The "Acc_block" is coupled to the embedded PowerPC processor via the APU (Auxiliary Processor Unit) interface. The principal reason for using the APU rather than connecting the hardware blocks to the CPU through the PLB bus is the superior bandwidth and lower latency between the PowerPC and the APU/FCM.

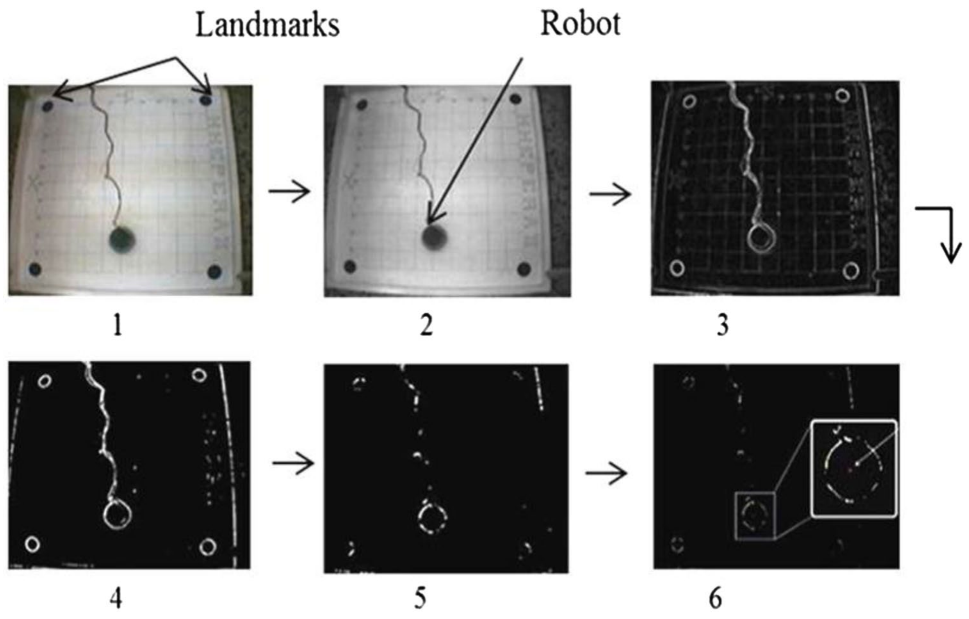


Fig. 4 The pipeline of image processing [25]

Fig. 5 Profiling Results [25]

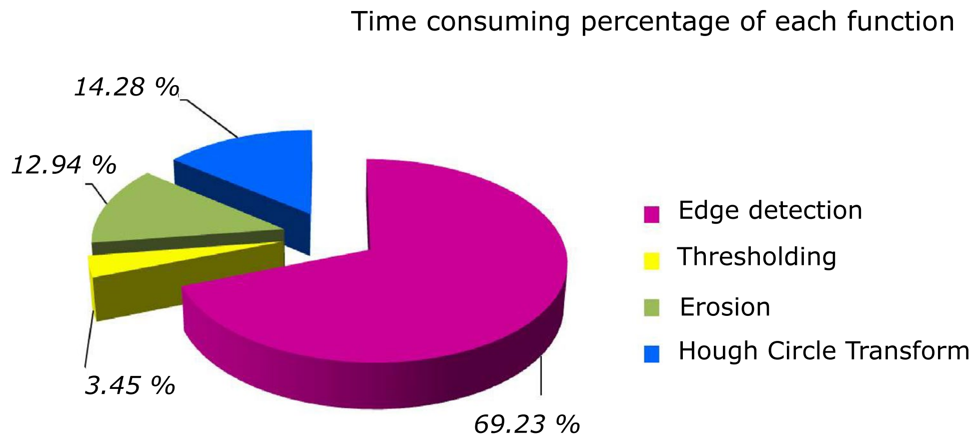


Table 4 Timing results for HW/SW implementation

Absolute localization algorithm	Results
Image frame size	640 * 480
Purely implementation: 400 MHz (s)	2.187
Purely implementation: 600 MHz (s)	1.445
+ FPU (s)	0.081
+ FPU + Acc block (s)	0.053
Acceleration factor	41.26

Table 5 Synthesis results of the architecture

Device		Virtex-5 xc5vfx70t	
Resource	Total	Usage	Usage percent
Device utilization summary			
Slice Registers	44.800	10.951	25
Slice LUTs	44.800	10.004	23
Number used as Logic	44.800	9.107	21
Number of occupied Slices	11.200	5.676	50
Total Memory used (KB)	5.328	2.610	48

An extra advantage is that the APU not dependent of the CPU to peripheral interface and consequently does not add a supplementary load to the PLB bus that the system involves for fast peripheral access.This function is implemented as a hardware accelerator called "Acc_Block". The "Acc_block" is coupled to the embedded PPC processor via the APU (Auxiliary Processor Unit) interface. The main reason for using the APU instead of connecting the hardware blocks to the CPU through the PLB bus is its superior bandwidth and faster communication with the PPC. An additional advantage is that the APU is independent of the

Fig. 6 The design architecture to obtained absolute localization

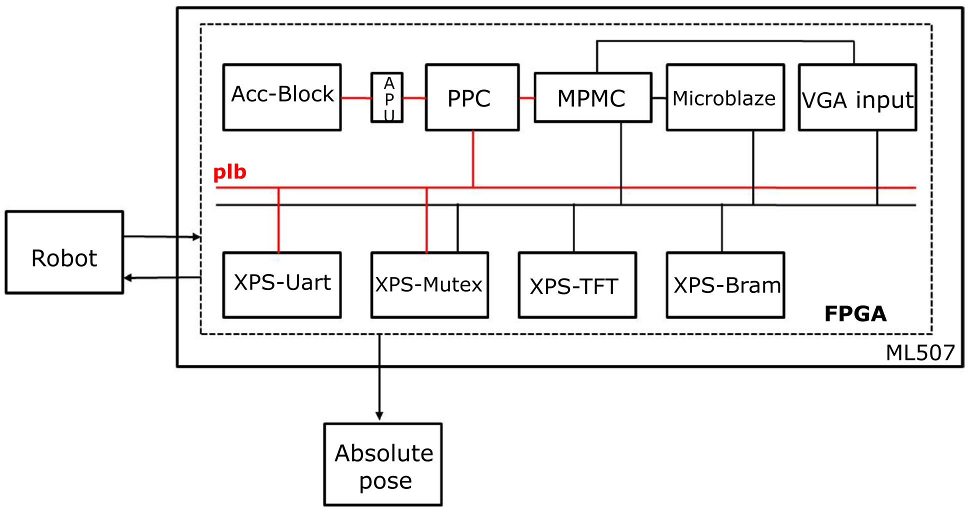


Table 6 Estimated Power and Energy of the system

Measures	Purely SW implementation	HW/SW implementation	Gain Factor
Power (W)	5.039	6.480	–
Energy/frame (j)	7.281	0.343	21.22

Table 7 FPGA slices occupation of both designs

FPGA slices occupation	
Relative localization	Absolute localization
3%	50%

CPU to peripheral interface and therefore does not add an extra load to the PLB bus, which is needed by the system for a fast peripheral access.

The processing time of the algorithm executed on the mixed architecture (PPC + "Acc_Block") when all the hardware enhancement techniques are used is about 0.053 s (see Table 4 for more details). The proposed HW/SW architecture gives acceleration up to 42 × compared to the pure SW implementation.

Figure 6 shows the architecture of the absolute localization system "archi_loc_abs"

The two processors share the external DDR memory through the MPMC module interface using an exclusion mechanism (XPS Mutex) to synchronize the access.

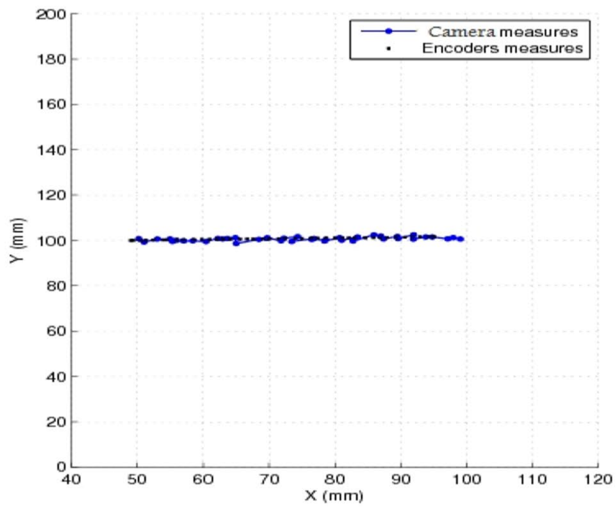
Tables 5 and 6 recapitulate the FPGA occupation and power consumption.

Table 7 displays the FPGA slices occupation for "archi_loc_rel" and "archi_loc_abs".

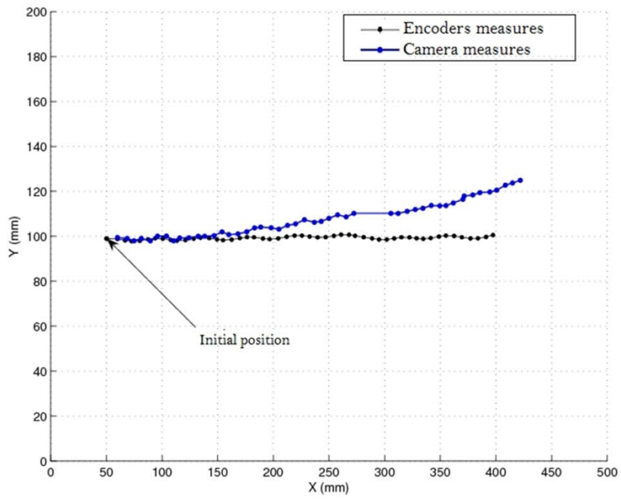
3.3 Experiments

The purpose of the achieved experiment is to simultaneously locate the robot in real time using both encoders and a camera. The results are presented in Figs. 7a, b and 8.

In Fig. 7a, the robot must follow a straight path for a distance of 50 mm. While in Fig. 7b, it must follow a straight path for a distance of 350 mm. Figure 7a shows that for a short trajectory, the two position measurements (odometer and camera) are very close. By cons, for a long



(a) Robot localization (short trajectory)



(b) Robot localization (long trajectory).

Fig. 7 Robot localization using both encoders and camera

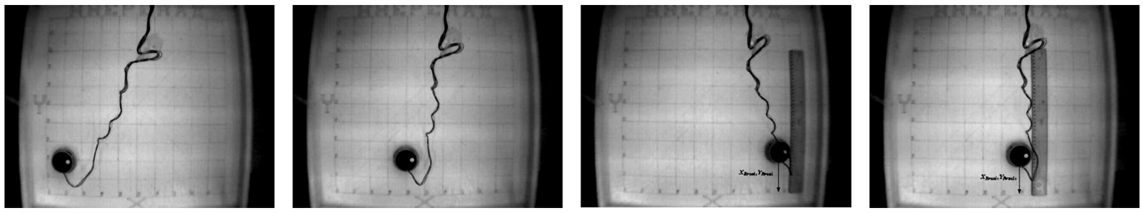


Fig. 8 Different robot real positions measured with a ruler

Table 8 Performance evaluating table

x_R(mm)/y_R(mm)	Measures							
	M1		M2		M3		M4	
	x_R	y_R	x_R	y_R	x_R	y_R	x_R	y_R
Methods								
Encoders	50	99	204.4	99.06	301.9	98.57	397.4	100.6
Camera	50	99	198.1	103.8	305.6	110.2	421.9	124.9
Ruler	50	100	200	104	306	115	424	125

trajectory, the error in the position becomes larger. This is the case of the experiment illustrated in Fig. 7b.

To quantify the errors of each method, we achieved four real positioning robot measures (*M*1, *M*2, *M*3 and *M*4) using a ruler as shown in Fig. 8. The real positioning robot is determined by measuring, by the ruler, its center position.

The results of this comparison are summarized in Table 8. The obtained values proved that the real measures are closer to those using a camera than those obtained using encoders.

3.4 Hybrid localization

The absolute location is crucial to correct the robot path measured by the odometer system. The hybrid localization consists in combining the relative position with the absolute one using a retiming point in order to correct the encoders' measures. The hardware architecture is the same presented in Fig. 6. If we consider the last point calculated by the camera at the finished journey as the retiming point, as shown in Fig. 9, the navigation error will be too

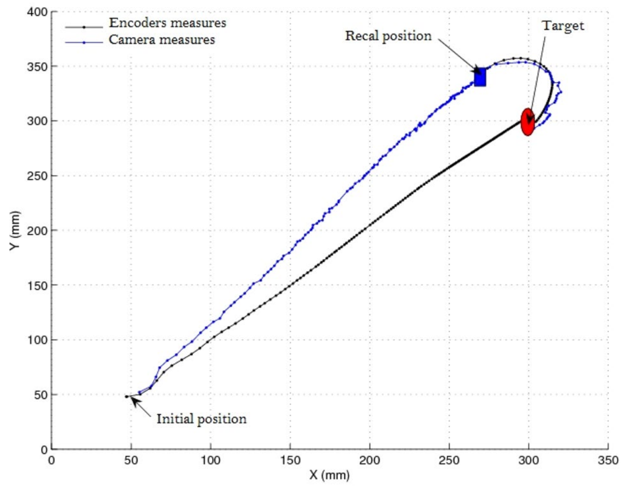


Fig. 9 Robot localization using retiming point

high as it is proportional to the travelled distance. In Fig. 9, the black curve represents the encoders' trajectory to be covered by the robot if there are no errors in encoders (relative localization) whereas the blue curve represents the robot real trajectory using camera measures (absolute localization). The error in this figure is 35 mm deflection along the x-axis and 40 mm deflection along the y-axis when the robot tries to reach the target T defined by (x_T = 300 mm, y_T = 300 mm), so about 10% error.

To reduce the error between the two positions, we have involved the absolute coordinates at each five frames and applied the correction on these points. This method allows speeding up the robot wheels velocity from 24 mm/s (the robot speed when applying the absolute localization on every frame) to 40 mm/s while reducing the navigation error: to a 12 mm deflection along x-axis and 17 mm along y-axis, i.e. about 6% error only.

The different localization techniques were firstly, independently designed in the ML507. In Sect. 3.5, we detailed how these three techniques can be used together in the ML507 according to the robot evolution conditions.

3.5 Dynamic reconfiguration of the localization system

3.5.1 The principle of the proposed approach

The dynamic reconfiguration can be used to enhance the FPGA resource utilization by time-sharing the reconfigurable resources among the different designs or among parts of a design. The internal FPGA architecture can be dynamically reconfigured at runtime to load on the fly when a new localization system is required. This mechanism allows optimizing the configured surface and managing the FPGA resources between different robot applications. The dynamic reconfiguration is achieved while the device is active: certain areas of the device can be reconfigured while other areas are operational without being affected by the reconfiguration.

When the hybrid localization technique is used, the relative localization is activated for four frames and the absolute localization is activated in the 5th frame. For the next

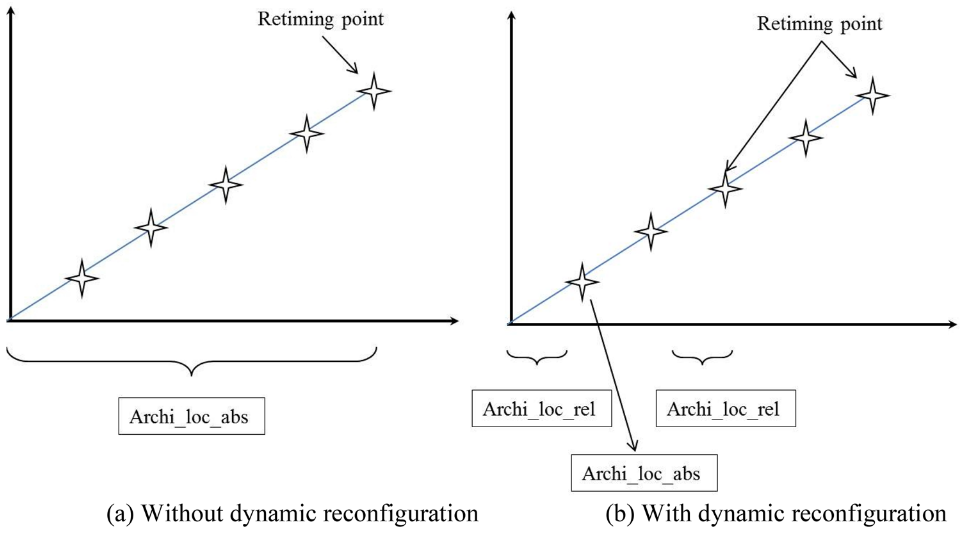


Fig. 10 The dynamic reconfiguration mechanism of the hybrid localization

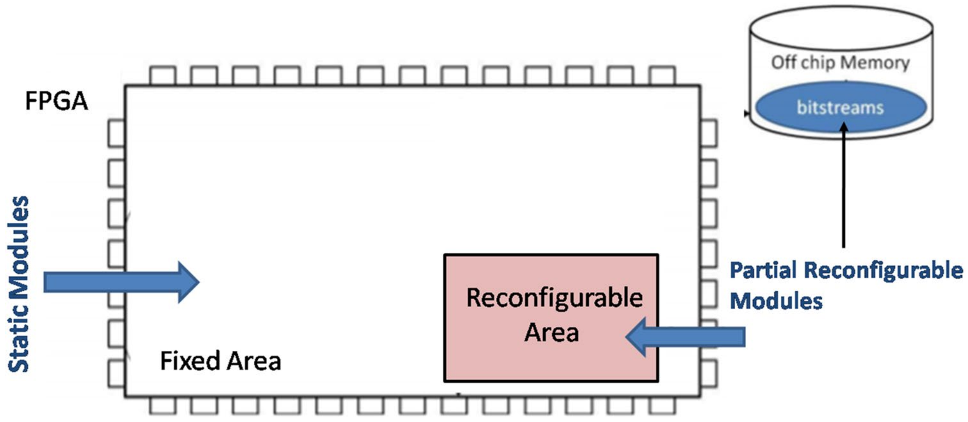


Fig. 11 Partial Reconfiguration FPGA showing static and dynamic regions

upcoming 4 frames, the relative localization is activated once again using the dynamic reconfiguration. Figure 10 shows how the dynamic reconfiguration mechanism is used in the hybrid localization technique.

In Fig. 10a, the *"archi_loc_abs"* is always active (50% of FPGA resources are allocated) without the dynamic reconfiguration. While, in Fig. 10b, during the relative localization, only 3% of FPGA resources are allocated. 50% of the FPGA resources are then liberated and used for other modules.

3.5.2 The concept of the dynamic reconfiguration

The logic in the FPGA design is split into two different types, static logic and reconfigurable logic. The white area of the FPGA block in Fig. 11 depicts static logic which does not change while the red portion represents reconfigurable logic that has been designed explicitly by the designer. This portion can be divided to many reconfigurable partitions. A series of modules are then assigned to these reconfigurable partitions (known as Partial Reconfigurable Modules). These modules are thereafter converted into partial bit files that can be, in run-time, switched in and out of the FPGA to map the suitable application functionalities into the associated partitions.

Figure 11 illustrates the key concepts for DPR model.

Our design invoked "archi_loc_rel" or "archi_loc_abs" to be used respectively for relative localization and absolute localization. Thus, two bitstreams are generated: a full bitstream that contains the static region which is designed by"archi_loc_rel" since the localization system starts by determining the relative localization, and a partial bitstream that contains the configuration of the partial reconfiguration partition which is"archi_loc_abs" loaded at every fifth frame.

To design reconfigurable systems, three approaches are possible: externally using JTAG or RS-232 interface, or internally using the auto-reconfiguration through the Internal Configuration Access Port (ICAP). We choose the auto-reconfiguration way which says that the FPGA

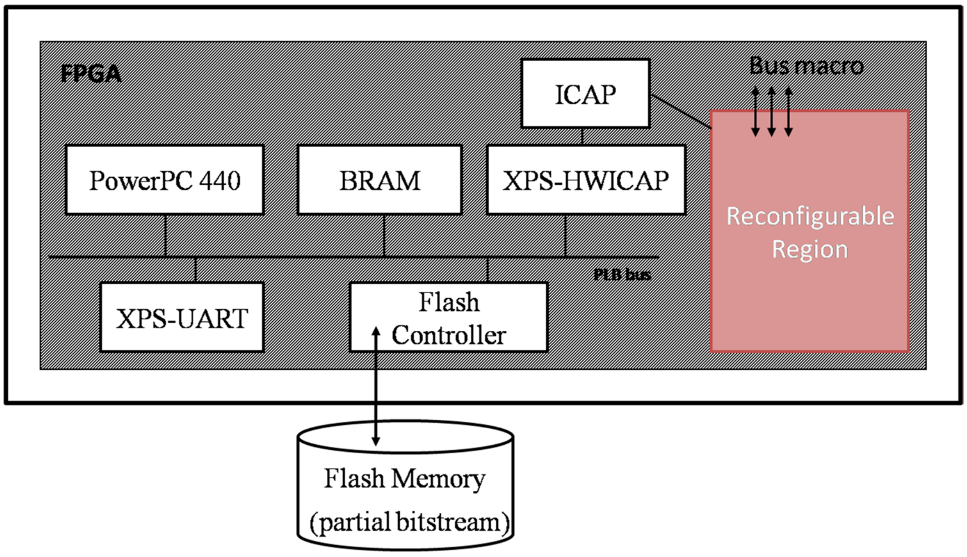


Fig. 12 The dynamic reconfiguration computing system

reconfigures itself to accomplish the purpose, either after predefined time length or in light of a status flag.

The ICAP module is used for the FPGA hardware dynamic reconfiguration. Xilinx provides an implementation of a core called XPS HWICAP that provides the interface necessary to transfer bitstreams files to and from the devices' configuration memory. An internal on-chip Block RAM can be utilized to store the recently read configuration data. The XPS HWICAP is connected to the PPC processor through the PLB bus. The system is connected to an external Flash memory (used to store the partial bitstream to carry out the reconfiguration) via a Flash controller that provides read/write access to the external memory. The communication between the static and dynamic regions is performed through interfaces called Bus Macros in Xilinx architectures.

Figure 12 shows the dynamic reconfiguration architecture.

3.5.3 System operation flow

The DR process is described according to the following steps:

- FPGA is programmed with initial configuration which is"archi_loc_rel" at power-up and starts execution.
- For each fifth frame, the PPC requests the partial bitstream "archi_loc_abs" from the compact flash and writes it in the PPC memory.
- PPC writes a word-by word to the configuration cache of the HWICAP module.
- The HWICAP BRAM is tested to verify if it has been fully charged.
- If Yes, the reconfiguration is carried out and all data contained in the HWICAP BRAM are copied to the FPGA through the internal BRAM memory.
- Checking if the FPGA reconfiguration has been done.

Figure 13 depicts the operation flow of the system.

3.5.4 Design flow

Figure 14 shows the methodology deployed in the implementation of dynamic reconfiguration strategy. The partial reconfiguration process demands separate Netlist files for the static (top level) design and for reconfigurable modules. Reconfigurable Modules are written in VHDL and are synthesized to a Netlist file using Xilinx Synthesis Technology (XST) whilst the static system Netlist is resulted from the MHS (Microprocessor Hardware Specification) description file. The obtained Netlist files are exported to PlanAhead tool that manages the details of creating and building the reconfigurable system. In PlanAhead tool the device region is partitioned into static and dynamic regions. Full and partial bitstreams are generated for different configuration.

As we conceive developing a complete robot navigation system with all its main activities, the required resources for absolute localization will be used by the remote control system. As the FPGA must instantaneously send sensors reading to the distant PC to control the robot in real time, we can use, for now and to validate the approach, this module when using relative localization. The remote control system is based on Ethernet connection. All sensors readings (images or values) are redirected and viewed from a distant PC using a web page.

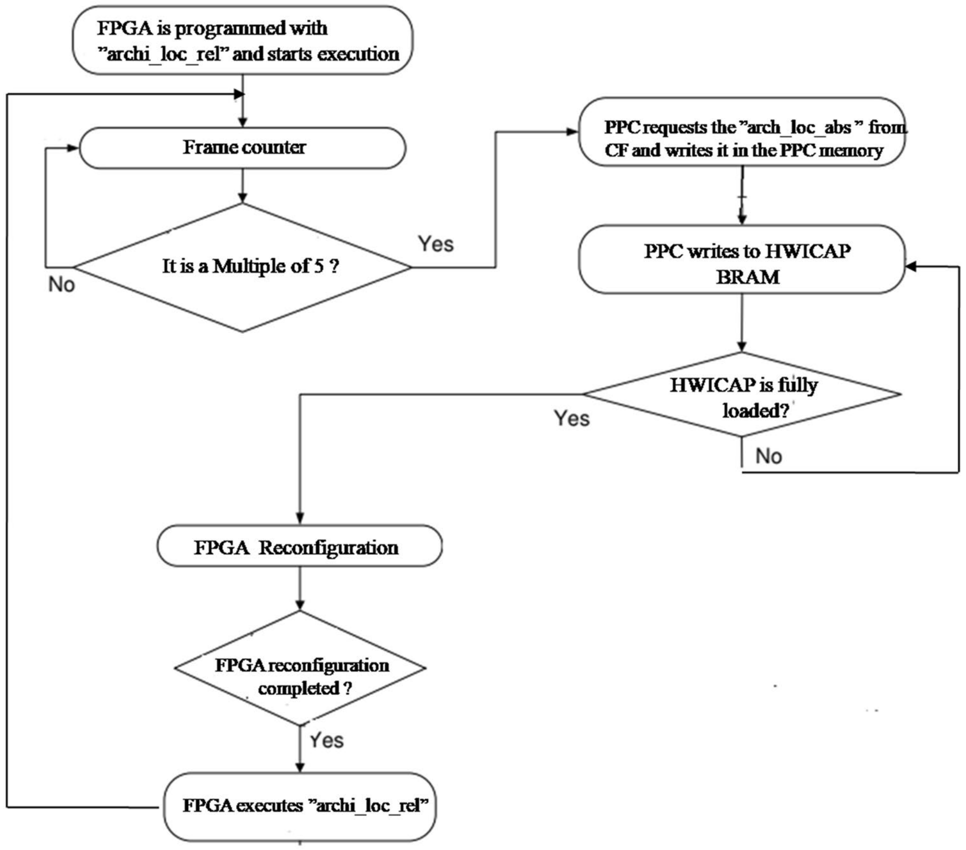

FPGA is programmed with "archi_loc_rel" and starts execution
Frame counter
It is a Multiple of 5 ?
No
Yes
PPC requests the "arch_loc_abs" from CF and writes it in the PPC memory
PPC writes to HWICAP BRAM
HWICAP is fully loaded?
Yes
No
FPGA Reconfiguration
FPGA reconfiguration completed ?
Yes
FPGA executes "archi_loc_rel"

Fig. 13 The operation flow of the system

4 The monitoring system

During the MR navigation, all sensors readings can be redirected and viewed from a distant PC (web client) using a web page.

The images provided by the ceiling camera and that fitted on the robot are available to the robot commander browser. The distant PC can thus visualize and monitor the robot in suspect places or if unpredictable obstacles are detected. It can also define a potential target coordinates and forward them to the installed web server on the FPGA.

The network image transfer is performed using the "output http" plugin of MJPEG streamer. HTTP output plugin or web server is the most important and the most used among output plugins. It can transport images to a web client navigator. The broadcast of the stream images via HTTP is very simple to implement. It's essentially to inform the client (the PC in our case), using a particular content type, that it will receive a series of files. As the client keeps the connection open, the broadcasting system continues to send images. Each new sent image replaces the previous image as soon as it is completely received. Clients must treat the different images as they arrive to avoid being crushed by the following images.

Figure 15 shows the HTTP header format.

The content type "multipart/x-mixed-replace" is the technique of sending data in the server push mode and streaming over HTTP. All parts of a mixed replace messages have the same semantic meaning, i.e., all have the same MIME type (images in our case). Nevertheless, each new sent image replaces the previous image as soon as it is completely received. Clients must treat the different images as they arrive to avoid being crushed by the following images.

This system connectivity is ensured through the Ethernet port of the Ml507 FPGA. The internet connection is used during the automatic mode to allow the user to track the robot.

The HTTP plugin must:

- Wait for a client connection (FireFox, Opera, Safari…) to serve robot images.
- Send the browser the headers that indicate the data type it will receive (HTTP headers).

The TCP socket does not report anything until the data is transferred otherwise it has to report the occurrence of an error.

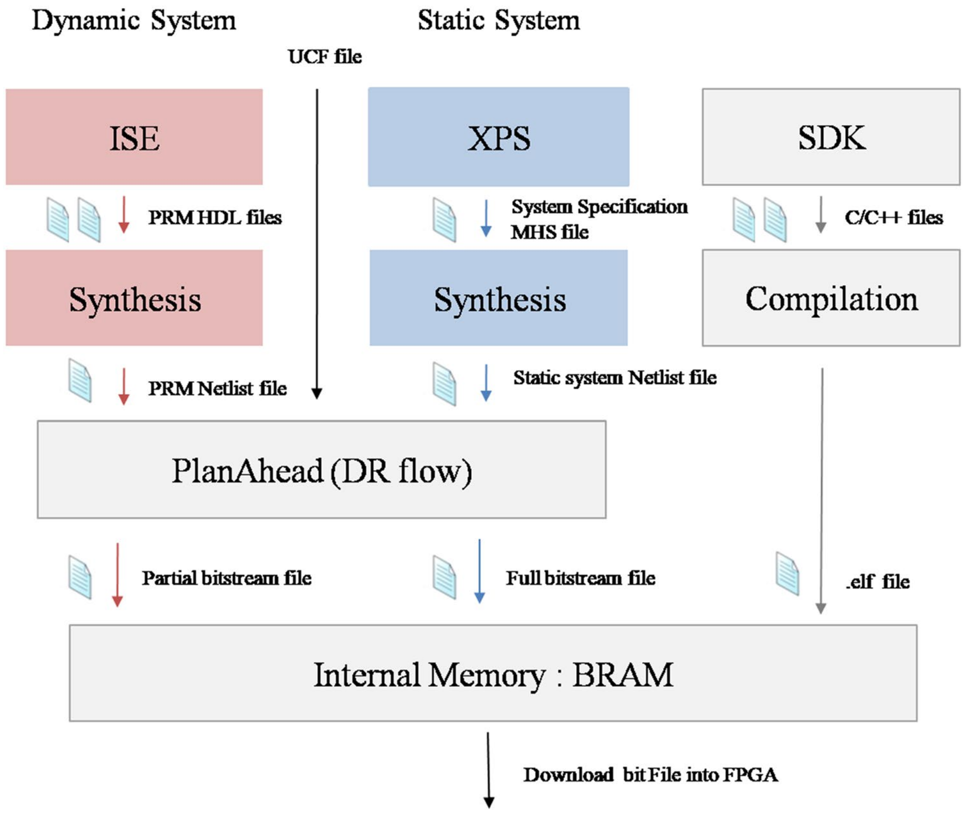


Fig. 14 Design Flow using XILINX Development Environment

Fig. 15 HTTP header, content type

```
HTTP/1.0 200
Expires: Thu, ddMonyyyyhh:mm:ss GMT
Content-type: multipart/x-mixed-replace; boundary=xstring

--xstring
Content-Type: image/jpeg
Content-Length: 97846
```

5 Conclusion

The implementation of a complete navigation system on a single board mounted on small autonomous robots has become possible thanks to the advances in the FPGA technology. The FPGA investigation is still not fully exploited in robotic vision applications. In this paper, we presented a flexible platform based on the FPGA technology, for studying and conceiving complete unified robot navigation system. This platform can be easily tuned to address the variable computing requirements of the robot localization application. We have independently implemented three localization techniques on the FPGA: relative, absolute and hybrid and we have benefited from the dynamic reconfiguration of the ML507 to change dynamically the localization system to satisfy energy/surface ratio and exploit the released resources to implement other robot tasks.

In ongoing works, we plan to study the complete FPGA robot navigation system that addresses the other major MR activities like obstacle avoidance and path planning.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Devi M, Kumar BA (2014) Accelerometer based direction controlled wheelchair using gesture technology. Int J Sci

Eng Technol 3(8): 1065–1070. https://pdfs.semanticscholar.org/1a04/f862f2843b67e1d45c2147f5414466824a9c.pdf
2. Jaiswar U, Repal S (2015) A review on real time breath processing based embedded wheelchair for quadriplegic people. Int J Innov Res Sci Eng Technol, pp 2319–8753. https://www.semanticscholar.org/paper/A-Review-on-Real-Time-Breath-Processing-Based-for-Jaiswar-Repal/37e190d5915a5ebde77b4d3f94002b6d58f3d496
3. Dos Santos MPS, Ferreira JAF (2014) Novel intelligent real-time position tracking system using fpga and fuzzy logic. ISA Trans 53(2):402-414. https://www.sciencedirect.com/science/article/abs/pii/S0019057813001468
4. Dong X, Su B, Jiang R (2018) Indoor robot localization combining feature clustering with wireless sensor network. EURASIP J Wirel Commun Netw 1: 175. https://link.springer.com/article/10.1186/s13638-018-1179-1
5. Storaasli OO (2008) High-performance mixed-precision linear solver for fpgas. IEEE Trans Comput 57(2). https://ieeexplore.ieee.org/document/4531732
6. Sanchez-Solano S, Cabrera AJ, Baturone I, Moreno-Velo FJ, Brox M (2007) Fpga implementation of embedded fuzzy controllers for robotic applications. IEEE Trans Ind Electron 54(4): 1937–1945. https://ieeexplore.ieee.org/document/4271570
7. Cruz S, Munoz DM, Conde M, Llanos CH, Borges GA (2013) Fpga implementation of a sequential extended kalman filter algorithm applied to mobile robotics localization problem. In: IEEE Fourth Latin Am Symp Circ Syst, pp 1–4. https://ieeexplore.ieee.org/document/6519021
8. Gugala K, Swietlicka A, Kolanowski K, Karon I, Majchrzycki M, Rybarczyk A (2013) Neural controller implementation in embedded system with use of fpga coprocessor. https://otik.uk.zcu.cz/handle/11025/11526
9. Chinnaaiah M, Priyanka G, Vani GD, Jyoti MA, Vennela K (2016) A new approach: an fpga based robot navigation for patrolling in service environment. In: International conference on research advances in integrated navigation systems (RAINS), pp 1–4. https://ieeexplore.ieee.org/document/7764422
10. Naji B, Abdelmoula C, Abbes K, Masmoudi M (2017) Design and test of a new development FPGA board for mobile robot research. Turk J Electr Eng Comput Sci 25(2):1483–1494. http://journals.tubitak.gov.tr/elektrik/issues/elk-17-25-2/elk-25-2-67-1510-18.pdf
11. Vennela K, Chinnaaiah M, Dubey S, Savithri S (2019) Implementation of mobile robot navigation mechanism using fpga: an edge detection-based approach, pp 215–222. https://link.springer.com/chapter/10.1007/978-981-13-2324-9_21
12. Magdum TD, P.C.B. (2017) Fpga based moving object tracking for indoor robot navigation. IOSR J VLSI Signal Process 7(5): 12–22. http://www.iosrjournals.org/iosr-jvlsi/papers/vol7-issue5/Version-1/B0705011222.pdf
13. Golen MB, Celik H, Yigit T (2018) Mobile robot control with FPGA. In: International engineering and natural sciences conference (IENSC 2018)
14. Ordóñez Cerezo J, Castillo Morales E, Canas Plaza JM (2019) Control system in open-source FPGA for a self-balancing robot. Electronics 8(2):198
15. Huang HC, Tao CW, Chuang CC, Xu JJ (2019) FPGA-based mechatronic design and real-time fuzzy control with computational intelligence optimization for Omni-Mecanum-wheeled autonomous vehicles. Electronics 8(11):13–28
16. Commuri S, Tadigotla V, Sliger L (2007) Task-based hardware reconfiguration in mobile robots using fpgas. J Intell Robot Syst 49(2): 111–134. https://link.springer.com/article/10.1007/s10846-007-9131-3
17. Griessl R, Herbrechtsmeier S, Porrmann M, Ruckert U (2011) A low-power vision processing platform for mobile robots. In: Workshop on computer vision on low-power reconfigurable architectures. https://www.semanticscholar.org/paper/A-Low-Power-Vision-Processing-Platform-for-Mobile-Griessl-Herbrechtsmeier/c815b2f006aa28c9fb75a8673d458cabb71b6091
18. Nava F, Sciuto D, Santambrogio MD, Herbrechtsmeier S, Porrmann M, Witkowski U, Rueckert U (2011) Applying dynamic reconfiguration in the mobile robotics domain: A case study on computer vision algorithms. ACM Trans Reconfigurable Technol Syst 4(3): 29 https://dl.acm.org/citation.cfm?id=2000841
19. Thakre MAK, Nagpur SDMP (2016) Reconfiguration of mobile robot. Int J Electron Comput Sci Eng 1(2): 161–165. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.224.8747&rep=rep1&type=pdf
20. K-team, 2001
21. Ogiso S, Kawagishi T, Mizutani K, Wakatsuki N, Zempo K (2015) Self-localization method for mobile robot using acoustic beacons. ROBOMECH J 2(1):12
22. Kim A, Eustice RM (2015) Active visual slam for robotic area coverage: theory and experiment. Int J Robot Res 34(4–5): 457–475. https://dl.acm.org/citation.cfm?id=2764727
23. Zhou JH, Lin HY (2011) A self-localization and path planning technique for mobile robot navigation. In: 9th world congress on intelligent control and automation WCICA, pp 694–699. https://ieeexplore.ieee.org/abstract/document/5970604
24. Silva J, Lau N, Neves AJ (2011) Localization techniques for autonomous mobile robots. Electronica e Telecomunicacones 5(2): 309-316. http://revistas.ua.pt/index.php/revdeti/article/view/2171
25. Ghorbel A, Jallouli M, Ben Amor N, Amouri L (2013) An FPGA based platform for real time robot localization. In: International conference on individual and collective behaviors in robotics (ICBR). IEEE, pp 56–61. https://ieeexplore.ieee.org/document/6729272
26. Amouri-Jmaiel L (2012) Contribution la commande et au pilotage réactif de robots mobiles roues. PhD thesis, Orlans