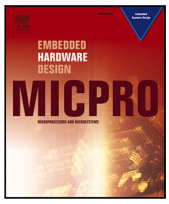
Microprocessors and Microsystems 113 (2025) 105143

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro



Open-source ROS-based simulation for verification of FPGA robotics applications

Rubén Nieto [a], Felipe Machado [a,b], Jesús Fernández-Conde [a,*], David Lobato [c], José M. Cañas [a]

[a] Rey Juan Carlos University, Spain
[b] Institute for Applied Microelectronics, University of Las Palmas de Gran Canaria, Spain
[c] JdeRobot Organization, Spain

ARTICLE INFO

Keywords:
Field-programmable Gate Array (FPGA)
Robotics
Open-source
Simulation
Robot Operating System (ROS)

ABSTRACT

FPGAs are increasingly incorporated in many high-end robotics applications, often involving computer vision and motor control. However, functional verification of FPGA designs is labor-intensive, time-consuming, and consequently expensive. Moreover, validation of complex systems, such as robots, poses even further challenges because neither the external interactions can be easily modeled with traditional testbenches nor the robot's response can be adequately observed and ascertained. This work presents a new methodology that validates the robot's behavior in a realistic simulated environment before transferring the design to the physical robot and the onboard FPGA. This methodology allows integral, fast, and flexible debugging cycles of robotics applications by integrating the functional simulation of the processing unit (FPGA) with the simulation of the robot, its environment, and their mutual interconnections. The Verilator simulation tool is used for fast Verilog/SystemVerilog verification and simulation. ROS, the standard robotics middleware, and Gazebo 3D robotics simulator are used for realistic robot simulation, including a robust physics engine. We have implemented several open-source software extensions to interconnect the Verilog circuit with the simulated ROS sensors and actuators. This methodology's utility and correctness have been assessed by developing a complete proof-of-concept FPGA-based robotics application in which a commercial robot follows a colored object using its onboard camera and differential drive motors. This work establishes the foundations for developing and testing complex robot FPGA-based modules more efficiently and flexibly.

1. Introduction

Robotics is one of the fastest-growing fields in science and technology, so innovation is decidedly sought after. Robots are helping humans not only in manufacturing industries but also in houses (vacuum cleaners), hospitals (DaVinci, surgery, rehabilitation), autonomous driving (Tesla, Waymo), inspection (drones), automated warehouses (Amazon), etc. In addition, the number of emerging robotics services is continuously growing.

Robotics is a challenging and demanding engineering field. Robustness and quick responsiveness are a must in this application domain. Robots handle a lot of heterogeneous data and usually demand real-time response. Robotic systems are complex because they involve multiple disciplines, such as software, electronics, mechanics, and control. Fortunately, there are robotic frameworks (simulators and tools) to help design and verify robotic systems, and the adoption of the middleware ROS as a de facto standard in the last decade has vastly improved software reuse and interoperability.

The utilization of Field-Programmable Gate Arrays (FPGAs) (also named reconfigurable computing or adaptive computing [1]) as the computing substrate opens many possibilities for augmenting performance and power efficiency while keeping economic costs low. FPGAs are increasingly incorporated into many high-end robotics applications as accelerators, primarily due to their parallel processing capabilities (vision systems, sensors, communications, etc.). In addition, sub-microsecond precision is a major benefit when these applications involve computer vision and/or motor control.

As [2] points out, despite all their advantages, FPGAs have not been fully embraced by the robotics community. The concurrent nature of the FPGA design, which is a clear advantage towards the robot performance, requires expertise in digital design and longer development times, adding a new layer of complexity to an already challenging

* Corresponding author.
E-mail address: jesus.fernandez@urjc.es (J. Fernández-Conde).

https://doi.org/10.1016/j.micpro.2025.105143
Received 14 August 2024; Received in revised form 26 November 2024; Accepted 3 February 2025
Available online 10 February 2025

design flow. In addition, porting the existing robotic applications into FPGAs is not straightforward. Developing robotic applications employing FPGAs is a rather complex endeavor, demanding diverse knowledge and tools, and using lower design abstraction levels than software. As stated in [3], many robotic applications are ideally suited for FPGAs; however, the robotics community is reluctant to adopt hardware acceleration because of the added complexity of the design and the lack of standardized programming models that span the software/hardware borderline.

Both domains, robotics and FPGA, use simulators for development. On the one hand, using robot simulators to test and debug robotics applications is common practice in software development in this field. General-purpose simulators such as Gazebo, CoppeliaSim, or Webots and specific simulators such as Carla (for autonomous driving) or AirSim (for drones) are widespread in the robotics community. They integrate robust physics engines such as ODE, Bullet, or Mujoco to achieve realistic emulations of robot movements and dynamics. They commonly include a rendering engine for realistic camera simulation and world visualization.

On the other hand, there are a plethora of proprietary and open-source digital simulators used for FPGAs, which allow visualizing and analyzing the waveforms of the signals of the circuit. Once the internal modules or the whole design is completed, they are simulated to verify that the circuit behavior is correct according to the specifications. This is an arduous and iterative task, requiring the design of comprehensive testbenches to provide adequate inputs for validating the circuit in different scenarios. In addition, interpreting the signals' waveforms to verify the circuit's correctness is a challenge on its own. This is particularly significant when modeling complex systems such as robots because it could be impractical not only to model the whole system using conventional testbenches, but also to assess the robot's behavior by inspecting the resulting waveforms. For example, it would be extremely costly to generate a conventional testbench that models a realistic interactive environment for the robot, containing obstacles, terrain slopes, variations in room illumination, collisions, etc. Moreover, the designer must assess whether the output motor control signals correspond to the desired robot trajectory.

As a consequence, verification of robotics FPGA-based applications poses a big challenge because two different knowledge fields and abstraction levels are combined. The present work (as part of the open-source FPGA-Robotics project[1]) proposes a methodology to design and verify robotics FPGA-based applications. This proposal connects the lower abstraction level of the digital FPGA design with the higher abstraction level of robotic simulators. It includes (a) the simulation of the FPGA processing unit, (b) the simulation of the physical robot and its environment, and (c) the interconnections between them. We have developed several software extensions (henceforth named *connectors*) that link the Verilog code with the robot sensors and actuators in ROS middleware. As far as the authors know, this is the first time a state-of-the-art robot simulator is connected to a functional FPGA simulator using open-source software for the whole toolchain.

The proposed methodology of combined simulation (now feasible in the current state of several technologies) presents certain advantages:

- It allows a holistic and interactive simulation of the robot's control Verilog code (including inertia, realistic movements, illumination changes, etc.). In this way, not only the basic functionality is checked, but also its high-level behavior. The robot's behavior is visually monitored in the simulator, whilst the FPGA internal processing is graphically represented and the timing diagram of the FPGA signals can be analyzed.
- It is safer because neither the robot nor the surrounding environment is damaged when debugging the FPGA Verilog source code that controls the robot.
- It provides fast debugging cycles as the Verilog code tests can be quickly performed in different scenarios and conditions. For instance, in various lighting conditions, with different robot mechanical properties (such as motor power, mass, or sensor positions), etc., receiving feedback instantly.
- It simplifies the debugging of FPGA-based complex robotic applications, as the tests do not require downloading into a board, setting up the configuration, or running the code on a physical robot.

In summary, this research work presents the following contributions:

1. The first contribution is a new methodology for the verification of FPGA-based robotics applications. It proposes intensive usage of simulation for testing and debugging before running the application on the physical robot and its onboard FPGA.
2. The second contribution is the integration of FPGA simulation (using Verilator) with robotics technologies such as ROS middleware and Gazebo 3D, by means of open-source software connectors that we have developed.
3. The third contribution is the experimental validation of the methodology by developing a vision-based robotics application as proof-of-concept (POC). In this POC, a TurtleBot2 robot follows a person using its onboard camera inside an indoor scenario. This application uses the open-source Verilog vision library presented at [4].

The rest of this paper is structured as follows: Section 2 presents a survey of related works. Section 3 details our combined simulation proposal for FPGA-based robotic applications. Section 4 is devoted to the experimental validation of the methodology by the implementation of a POC. Section 5 discusses the main benefits of the proposed methodology. Finally, Section 6 concludes the paper.

2. Related work

2.1. FPGA development

FPGA development has been typically performed using Hardware Description Languages (HDL), such as Verilog and VHDL. These languages allow describing digital circuits at the Register Transfer Level (RTL), which can be synthesized and implemented on an FPGA using Electronic Design Automation (EDA) tools. Traditionally, these EDA tools have been proprietary, but many open-source projects have recently been sprouting around FPGA development, such as Project IceStorm [5], Icestudio,[2] and F4PGA[3] (formerly Symbiflow).

Due to its lower abstraction level, describing circuits at RTL is time-consuming, requires experience in hardware design, and demands deep verification [6,7]. To overcome these limitations, High-Level Synthesis (HLS) tools have arisen, allowing the generation of RTL circuits from designs described at higher abstraction levels. Nevertheless, both Verilog and VHDL are still widely used by the industry [8] and the output of most of the HLS tools is RTL code that has to be further synthesized into FPGA. Therefore, verification at the RTL level is still a requirement.

As it has already been stated, verification of FPGA designs is challenging and critical [7], taking an average of 50% of the total project development time [8]. Functional simulation is used to verify the correctness of the circuit. Simulations can be performed at different levels, such as event-driven and cycle-accurate, which are the most used in digital circuit design. Event-driven simulators provide information on the signal events within a clock cycle, as opposed to cycle-accurate simulators that only provide information on the signals' state at clock

[1] https://github.com/JdeRobot/FPGA-robotics

[2] https://icestudio.io/

[3] https://f4pga.org/

cycles. Therefore, cycle-accurate simulators are less detailed but considerably faster. There are open-source and proprietary simulators. An overview of Verilog HDL simulator technology is given in [9]. RTL simulators are computationally intensive, and there have been efforts to optimize their performance latterly [10,11].

To reduce the time spent in verification, testbenches are usually described at a higher abstraction level than the RTL Design Under Test (DUT). Although VHDL and Verilog allow describing higher-level non-synthesizable testbenches, there is an increasing interest in other languages such as SystemVerilog, or even software-oriented languages such as Python (for example using cocotb). Using software languages such as Python makes verification easier because it is a commonly known language, plenty of libraries are available and a higher degree of abstraction is possible.

2.2. Robotics software development

Robots have traditionally relied on Central Processing Units (CPUs) with generic processors as the mainstream. The typical technique for achieving more computation power consists of adding more cores and processors to the robotic system in order to perform parallel computing. In the last few years, GPUs have also been integrated into robots as computing substrate, mainly to speed up the computer-vision operations (with OpenCV, for instance) and deep-learning-based software (with Pytorch or TensorFlow, for example), both at inference time and at training time. Regarding programming languages, robots can be programmed in many languages, but C++ and Python are the most commonly used for robotics applications.

ROS middleware [12,13], fostered by OpenRobotics, has become the de facto standard in robot programming. It is open source and has a distributed nature. With ROS, robotics applications are typically distributed among several concurrent nodes, which communicate between them, sending messages through a protocol and standardized ROS Interfaces (topics, services, actions). The ROS nodes may run in different CPU cores and even different CPUs. In recent years, a significant refactoring has created ROS2 [14] over the long experience with ROS-1. ROS2 uses better and safer communication middleware (DDS), and it is Windows-compatible.

ROS has a large worldwide community. There exist ROS drivers for many robots, sensors, and actuators. In addition, it provides an ecosystem with many tools and reusable libraries or reusable nodes for robot control, motion planning, perception, etc. For instance, the ROS Navigation stack solves general mobile robots' movement, planning, and self-localization; MoveIt is for industrial robots; and PX4 and ArduPilot are for drones. Also, there is a set of development tools that improve the robot programming process: ROSbags for recording and replaying robot data, rViz for visualization, etc.

Other useful tools when developing reliable robotics applications are robot simulators [15]. They provide the initial verification of software designs and allow applications' testing without damaging any physical robot. Typically, the simulation phase extends during a large debugging and testing period before testing and running the application on the physical robot. Robot simulators shorten the development iterations and speed up the testing, resulting in more robust applications at earlier stages.

Most widespread simulators support ROS applications directly or through bridges. Gazebo is a popular general-purpose, open-source 3D robot simulator fostered by OpenRobotics. It is fully integrated with ROS. In addition, other robot simulators such as Webots, CoppeliaSim, NVIDIA Isaac-sim (for AI-based robots), or even CARLA (the reference simulator for autonomous driving) supports ROS applications. Many simulators include high-fidelity simulations with realistic 3D scenarios and noise in sensors and actuators. With robotics applications, formal verification of the source code is not enough; the effects of such noises and movement dynamics (inertia, masses, friction, etc.) must be considered in the simulation to perform useful testing and reduce the gap between simulation and reality.

2.3. FPGAs in robotics

The robotic applications may benefit from parallelizable and energy-efficient devices such as GPUs and FPGAs. Nice and complete surveys of FPGA-based computing in robotics are available [16–19]. Some works explore the general integration of FPGA processing into ROS systems [20] or focus on the execution of the ROS communication protocols on FPGAs. But FPGAs are mainly used as accelerators of some algorithms in several *robotics tasks* [21] such as sensor data processing, vision-based perception, localization and planning or control. There are illustrative examples of FPGAs used in several *robotics application domains*, including drones [22–24], industrial robots [25], autonomous driving [26] and walking robots [27].

An interesting way of combining ROS with FPGAs is proposed in [28], in which a ROS node implemented in software accesses the hardware component, i.e., the accelerator, via a software wrapper. Communication within the ROS network is completely handled in software, and whenever acceleration is needed, only the payload of the ROS message is transmitted to the hardware component. A more recent relevant proposal in the same line is the ReconROS framework [29], which can map complete ROS2 nodes into hardware for acceleration. It has been recently extended and improved with the ReconROS executor [30].

The FPGA-ROS methodology [31] proposes a generic interface for all hardware FPGA modules, following the modular design of ROS. It leads to fully customizable messages to exchange data internally or with other parts of the distributed systems.

Some relevant works focus on accelerating the ROS communication protocol inside the FPGA code. For instance, a hardware ROS-compliant FPGA component was created in [32], separating the registration part (XMLRPC) and data communication part (TCPROS) of Publish/Subscribe messaging. It was reviewed and extended in [33]. A recent work [34] proposes a communication data distribution service for hardware-mapped ROS2 nodes (as in [29]), which provides speedups of up to 13.34 and jitter reduction by two orders of magnitude.

Regarding the acceleration of robotics tasks and algorithms, one example in sensor readings is the FPGA-based 3D LIDAR sensor built with multiple inexpensive 2D RPLidar A1, which are rotated via a servo motor and their signals combined with an FPGA board [35]. The fast FPGA processing is also useful in vision-based perception, as in [36] where they are used for a real-time sift matcher and RANSAC algorithm. Another interesting example is [37], where they are used to accelerate the road-image processing and traffic light recognition using the HOG feature and SVM classifier. As a result, traffic light recognition with FPGA is 270 times faster than those only with CPU. In ReconfROS [38,39], a hardware-software codesign solves the vision-based path following application for an Unmanned Guided Vehicle. An embedded CPU processor cooperates with an FPGA through a shared memory. The CPU runs the networking and the FPGA runs the image processing and control algorithms synthesized from C-code. They achieve a significant boost in performance (x3) and a reduction (1/15) in energy per frame. In addition, AI processing inside robots has also been optimized with FPGAs [40].

Robot localization and SLAM algorithms, in particular vision-based ones, have been accelerated with FPGAs as well [41–44]. In addition, FPGAs have also optimized robotics decision-making tasks such as Control [45–48] and Planning [49–51]. A relevant example is [52], where the authors have implemented the classic A* path planning algorithm in Verilog RTL language and into a Xilinx Kintex-7a FPGA for real-time performance. It achieves 37–75 times performance enhancement relative to software implementation.

2.4. Verification of FPGA-based robotic applications

The review of the previous section showed an abundance of proposals that include FPGAs in robotic applications. Unfortunately, most of them do not provide a clear explanation of how the designs have been verified.

Nevertheless, some works do provide information about the verification methodology. These methodologies can be classified into four approaches:

1. High-level simulation: if the FPGA design is implemented using High-Level Synthesis (HLS) (Section 2.1), the functionality has been described in a high-level language, such as C/C++. As a consequence, this description can be simulated in a robot software simulator, as in [21]. This approach does not verify the resulting RTL description, but the software model of that description, which can lead to divergences in the behavior or the timing performance.
2. FPGA module verification: when the FPGA is used as an accelerator for a specific robot's task, the verification can be performed just at the FPGA module level. For example, Podlubne et al. [20] substituted software components with FPGA-based ones while retaining the same ROS communication interface. Thus, they simulated the FPGA components to verify that the interfaces were correct. Another example is [36], in which the authors developed a robotic vision module. They verified the FPGA algorithm through simulation and later validated it on an FPGA board. This was performed independently of the robot application. Other proposals perform a similar approach, such as [52] for real-time path planning. This approach is necessary for developing any medium complexity FPGA design because it allows observing the internal behavior of the system. However, it has limitations due to the difficulty of modeling a realistic testbench for a complex system, such as in robots.
3. Hardware in the loop simulation: the verification is performed by connecting the implemented FPGA design to a computer with a running robotic simulator [30,53,54]. This is an interesting approach because the FPGA module is simulated in a realistic environment. In addition, the FPGA is running in real-time, although there could be some delays due to the computer connections.
4. Robot validation: in this approach, the FPGA design is tested in the physical robot [44,46–48]. This approach requires a safe environment and the causes of malfunctioning can be difficult to assess.

These four approaches should be used in conjunction when possible, especially in complex designs, because they provide benefits in the different stages of the design. For example, the first step would be to perform a high-level simulation to check the feasibility of the algorithms. Then, verification of the individual FPGA modules is performed by checking the timing diagrams and interfaces. Next, a HIL simulation could test the robot in a simulated environment. The final stage would validate the design in a real robot.

Nonetheless, we consider that it would be highly beneficial to add a verification stage in which the low-level FPGA simulation is connected to a robotic simulator. This approach allows testing the FPGA design in a realistic environment while observing the internal behavior of the signals, which cannot be simultaneously obtained from any of the previous approaches.

In this work we present a methodology to verify FPGA-based robotics applications by interconnecting the simulations of the FPGA, the robot, and the physical environment using open-source software and tools. As opposed to traditional methods, our approach not only validates the basic functionality of the FPGA application (signal processing) but also simulates the robot's high-level behavior dynamically.

3. Combined simulation for verification of FPGA-based robotics applications

In order to validate the combined simulation methodology proposed in this work, we have set up a simulation system that integrates the functional simulation of the processing unit (FPGA) with the simulation of the robot and its environment. This versatile simulation-based system, enhanced by the collaborative capabilities of open-source tools such as OpenCV, ROS, and Gazebo, offers an alternative to traditional hardware design testbenches for FPGA robotics applications.

In this simulation scheme, complex FPGA inputs (such as images) are generated and fed to the simulated FPGA, supporting the interaction with the generated FPGA outputs (such as motor commands) to provide fast updated feedback to the FPGA design. This contrasts with real-world testing where, for example, visual inputs are always different. It also supports the creation of different scenarios with obstacles, slope, uncertainty, terrain, weight, etc. that can change the performance of the FPGA design. This allows the development of robust and efficient systems, providing the ability to test and debug the robot in a safe and controlled environment. In addition, it is possible to debug the different variables that interact with the robot in order to understand its response more clearly and visually.

3.1. System architecture

Fig. 1 shows a block diagram of the system. The proposed combined simulation system (light gray rectangle above) can be connected to real and simulated robots. It can also be connected to a Display to show its GUI. The bottom part of the figure shows the simulated robot (left) with ROS drivers for its simulated sensors and actuators, and alternatively the real robot (right) with the ROS drivers for its sensors and actuators.

The combined simulation system consists of two parts:

1. Verilated FPGA Modules: developed in C++, they rely on the open-source simulation tool Verilator to simulate circuits described in Verilog or SystemVerilog. Verilator transforms a Verilog HDL module into an optimized C++ model (Verilated module) that can be compiled with any C++, such as GCC. The compiled module can be instantiated in a C++ wrapper that will provide the stimuli to the module when executed, thus carrying out the simulation.
2. Connectors for robot sensors and actuators: developed in C++, they link the Verilog modules with the ROS drivers of the robot's sensors and actuators. Connectors are detailed in the next section.

The use of C++ as the base language makes Verilator perfectly suitable to be combined with other open-source tools and libraries popular in robotics such as OpenCV, ROS, and Gazebo. Consequently, complete and integrated systems in the field of robotics and computer vision can be developed in Verilog and tested with simulated FPGAs in Verilator and simulated robots in Gazebo.

The robotics middleware chosen for robotics applications in this system is ROS, mainly because it is the de facto standard and it provides a wide collection of drivers for many robots.

In the case of real robots, some drivers are needed to connect to the physical sensors to get the sensor readings or connect to the physical actuators to send them commands. These drivers are ROS nodes, used to provide access to robot sensors/actuators through ROS *topics*, which can be seen as communication buses over which different software pieces exchange messages.

In the case of simulated robots, Gazebo[4] is the reference simulator. It is a high-quality open-source software maintained by the Open Source Robotics Foundation (OSRF) and widely accepted in the international

[4] http://gazebosim.org

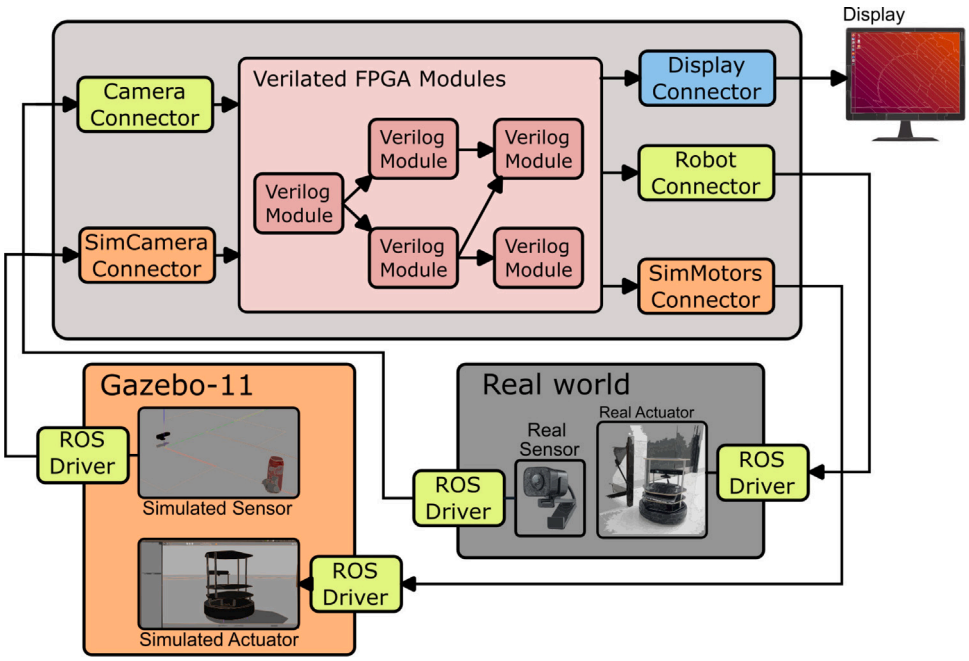


Fig. 1. Conceptual diagram of the simulation system including simulated FPGA and its interaction with the robot sensors and actuators (real or simulated).

robotics community. The Gazebo simulator materializes the behavior of the robot in a virtual scenario. The drivers are plugins that run inside the simulator itself and provide access to the sensors and actuators of the simulated robots through the same ROS topics as the equivalent real ones. In order to start a world simulation, a configuration file that determines the scenario, the robots involved, etc., is created and used. Gazebo allows a local viewer to observe the evolution of the simulated world, including the behavior of the programmed robot, also allowing interaction with the scene by adding certain elements at runtime, stopping, re-launching the simulation, etc.

The proposed combined simulation system (Fig. 1) provides a full development and testing toolchain for FPGA-based robotics applications developed in Verilog.

3.2. Sensor and actuator connectors

As it has been explained, the Verilated code is a cycle-accurate C++ model of the FPGA design. In order to include this model in the proposed simulation scheme, we have developed several software connectors to link the Verilated model to the robot's sensors and actuators, real or simulated. These software connectors make the sensor readings available to the Verilated modules and send the outputs to the appropriate actuator to extend the usefulness of the simulated system. Therefore, a Hardware Abstraction Layer (HAL) upon which the robotics applications may be built is provided.

The novelty of these connectors lies in their ability to operate in heterogeneous environments, ensuring data exchange and synchronization between the FPGA simulation and the robotic middleware. Key considerations during their development included:

- Low latency: Ensuring low-latency communication between the Verilated simulation and ROS, critical for maintaining robotic control loops.
- Scalability: Designing connectors adaptable to various types of sensors and actuators, whether physical or simulated, allowing easy reuse in different robotic applications.
- Consistency: Providing identical interfaces for real and simulated systems by adhering to ROS topic conventions, enabling seamless transitions between development stages.

These connectors are parameterizable to fit the specific requirements of each simulation. For instance, the Verilated code running the FPGA simulation can be configured to emulate different hardware peripherals using Verilog code, and the connectors adapt accordingly to expose these peripherals through ROS topics using C++ code.

The integration is achieved by embedding these connectors directly into the system architecture. ROS nodes publish sensor data generated by the FPGA or receive actuator commands and forward them to the FPGA simulation. These connectors operate at the middleware level, bridging the software and hardware domains to create a unified combined simulation system.

3.2.1. SimCamera connector

The SimCamera connector, located in this GitHub repository,[5] allows the Verilated model to receive a full frame stream from a simulated camera. This functionality is performed using ROS and Gazebo. A simplified illustrative diagram of the design is shown in Fig. 2.

This connector provides the Verilated module with the camera footage. The SimCamera connector uses ROS to handle all the communication between the C++ application that instantiates the Verilated model and the simulated camera in Gazebo, where the image data comes from. This is done through ROS topics.

Gazebo supports SDF files to describe the simulation to be loaded. An SDF file defines the world, the robot's characteristics, and what plugins to load. A `camera.world` SDF file[6] was developed to simulate the robot (consisting of just a camera) and the world, and to load a camera plugin that publishes the generated images to the ROS topic "/image_raw".

The SimCamera connector subscribes to the same topic and instantiates a callback function to handle new image data received from the simulated world. These data are then passed to the Verilated model for image processing. The following code shows a part of the code developed for this connector, specifically the aforementioned callback function and how the subscription to the ROS topic is made.

[5] https://github.com/JdeRobot/FPGA-robotics/tree/master/sim_fpga/examples/poc/example5

[6] https://github.com/JdeRobot/FPGA-robotics/tree/master/sim_fpga/examples/poc/example5/worlds

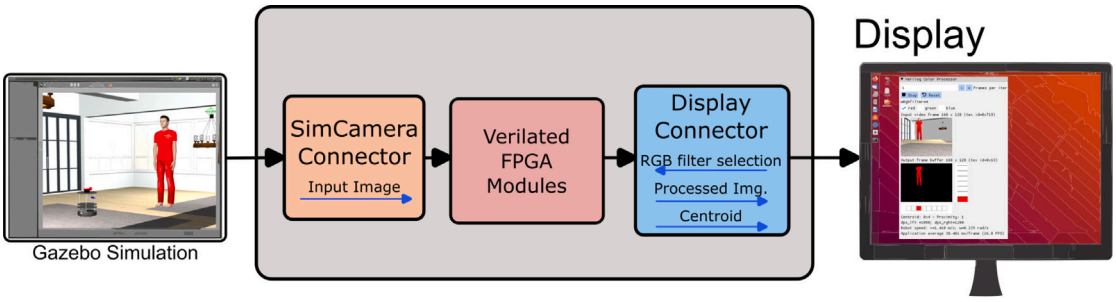


Fig. 2. SimCamera block diagram enables the Verilated model to obtain a complete frame stream from a simulated camera.

```
//ROS Integration
ros::init(argc, argv, "image_listener");
ros::NodeHandle nh;

image_transport::ImageTransport it(nh);
image_transport::Subscriber sub = it.
  subscribe("image_raw", 1, imageCallback);
```

Code 1 C++ code for ROS Integration.

```
void imageCallback(const sensor_msgs::
  ImageConstPtr& msg) {
 try {
  input_feed = cv_bridge::toCvCopy(msg, "
  bgr8")->image;
 } catch (cv_bridge::Exception& e) {
  ROS_ERROR("Couldnotconvertfrom'%s'to'bgr8
  '.", msg->encoding.c_str());
 }
}
```

Code 2 C++ code to handle the image data received from ROS topic for Verilator modules.

3.2.2. SimMotors connector

The SimMotors connector supports the control of a differential drive simulated robot from the Verilog application code. It also implements the required adaptation to integrate the outputs between the Verilated Model and the simulated world. For instance, the typical FPGA output from a Verilog application for the motor control of a differential drive robot is the rotation speed for the left wheel and right wheel in degrees per second (DPS). However, the ROS topic for motion control is composed of linear speed (V) and angular speed (W) of the whole robot. This connector carries out the needed conversion between the different formats.

The C++ SimMotor connector transmits movement commands to the simulated robot using the ROS topic `/cmd_vel`. This topic handles `geometry_msgs/twist`[7] messages consisting of two 3-dimensional vectors representing linear and angular velocities (V and W respectively).

A simulation model for the Turtlebot 2 robot was used specifically for Gazebo (Fig. 3).

3.2.3. Graphical user interface and Display connector

As an additional debugging tool, a specific Display connector was developed to show text values, raw images, filtered images, centroid values in columns, and proximity values. As the SimMotors connector, it can be linked to the output of other Verilated modules. The final GUI is shown in Fig. 4, which provides the robot camera view of the simulated world and the filtered image produced by a Verilated module. In this case, the red filter is selected, so every detected red pixel is kept in the image and the rest is changed to black. The LEDs on the right indicate the proximity to the object, as the amount of red pixels in the example is low, it is considered to be far away. On the other hand, the LEDs at the bottom of the image indicate how centered the target is with regard to the horizontal plane of the camera; in the example, the object is slightly to the left. In addition, the GUI also includes the angular velocity of both wheels in degrees per second (DPS), and the robot's linear (V) and angular (W) speeds.

The Display connector provides invaluable visual information regarding the internal processing of the Verilog module, such as how well the color filter is performing, and if the target is being correctly detected. This would be very difficult to assess in a typical Verilog testbench by just observing waveforms; or when using either Hardware in the Loop simulation or real robot validation, in which access to internal signals is not feasible.

3.3. Simulation of Verilog modules

The robotics application in the proposed simulation scheme is a combination of several Verilog modules. Verilator[8] is the selected tool to simulate the operation of those modules. It replaces the physical FPGA with behavioral models of Verilog modules in C++ or SystemC.

For a specified class prefix, Verilator will output a `prefix.h` header file that defines a class named prefix that represents the generated model to be instantiated. It also creates a `prefix.cpp` file, along with additional header and implementation files. This model class defines the interface of the Verilated model and the output of the process will also contain a `prefix.mk` file that can be used with "Make" to build a static library with all the required objects.

The generated model class file manages all internal states necessary for the model and provides an interface for interaction. This includes exposing top-level IO ports, public module instances, and the root of the design hierarchy. In C++ output mode, the user is responsible for writing a C++ wrapper and main loop for the simulation, which involves instantiating the Model class and linking to the Verilated model.

Verilated models implement the internal logic of the hardware in software, allowing more efficient debugging using tools such as (GNU Debugger) or the simple `fprint` trace. In addition, results can be displayed more visually, and Verilog modules can produce VCD output files to aid in debugging using waveform visualization software such as GTKWave.[9]

Verilator's performance is considerably higher than conventional simulators because the Verilated module is optimized to perform a

[7] http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html

[8] https://www.veripool.org/verilator/

[9] http://gtkwave.sourceforge.net/

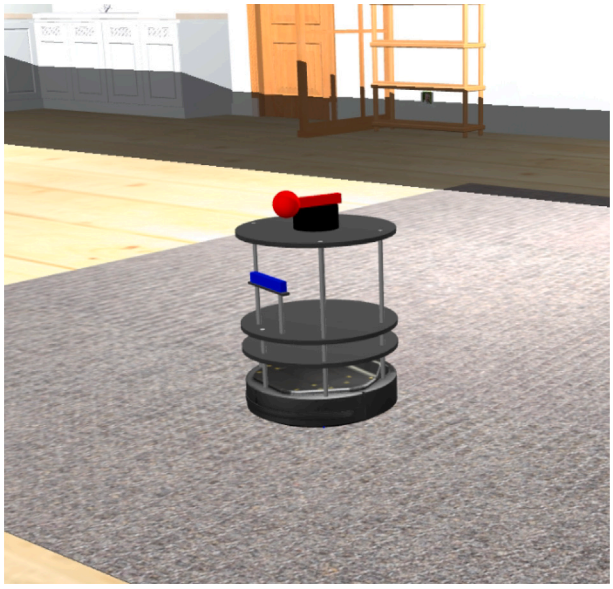


Fig. 3. Differential Drive Robot in Gazebo equipped with a camera (shown in blue); the identifier for the direction of movement is depicted in red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

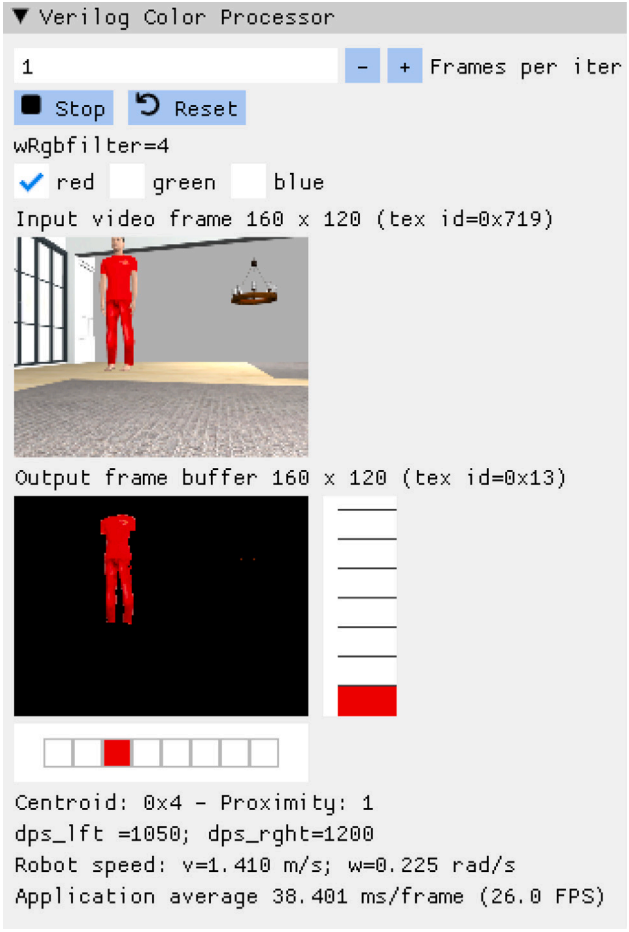


Fig. 4. The GUI displays the robot's camera view of the simulated world and a filtered image. The red filter highlights detected red pixels, with LEDs indicating object proximity and alignment. Angular velocity and linear/angular speeds are also shown. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

cycle-accurate simulation, instead of an event-driven simulation. As pointed out in Section 2.1, cycle-accurate simulations are less detailed but substantially faster. Cycle-accurate simulation should be enough if the model is synchronous and the setup and hold requirements are met, which can be verified in lower-level verification.

A key advantage of working with Verilog modules is their reusability, regardless of the platform where they will be implemented. Therefore, the advantage of the Verilator simulator is that it allows the behavioral output of the design, which facilitates its adaptation to different development environments and implementation platforms.

A library of Verilog modules is being developed for robotics and vision applications.[10] They speed up the development integrating some of them in new applications. These Verilog modules aim to control the movement of robotic platforms or to process camera signals. For vision applications, they provide real-time image processing, such as object detection and tracking. The main advantage is that the reusability and adaptability of these modules make them ideal for applications where a high degree of flexibility and performance is required.

4. Experimental validation

Using the proposed simulation scheme, a complete POC has been developed for the verification, testing, and fine-tuning of a non-trivial vision-based robotics application in Verilog. This POC validates both the implementation of the methodology itself and the correct behavior of the created robotics application.

The source code for this POC is open and can be found here.[11] The development process demonstrates some of the advantages of the proposed simulation methodology: it allows fast verification, easy fine-tuning of the Verilog source code, and agile tests of both the application modules and the whole application.

4.1. Robot and scenario

The POC testing and evaluation were conducted using a TurtleBot2 robot in an indoor scenario[12] simulated using Gazebo-11 and with ROS1-Noetic middleware. During the simulation, a person dressed in red was included and moved around the simulated environment.

This scenario was used to simulate realistic conditions for robot operation, thus allowing a fine-grained evaluation of its performance in a controlled but dynamic environment. The presence of a person dressed in red was used as a specific test case to evaluate the robot's ability to detect and respond to moving objects in its environment. This test approach allowed us to evaluate the effectiveness of the FPGA algorithms for object detection and tracking and validate the connectors used, as well as, the ROS communication with the robot. Fig. 5 shows a screenshot of the simulated environment running in Gazebo, showing the person dressed in red and the TurtleBot2 robot.

4.2. Visual follow person robotics application

Fig. 6 shows a simplified diagram with the main blocks of this robotics application POC. The system includes an instance of the Verilated Modules, a GUI to display the input and output images using the GUI connector, the SimCamera connector, the SimMotors connector, and all the necessary logic to communicate with the Gazebo simulated world.

The SIM FPGA-Robotics[13] repository includes several functional Verilog modules and a Graphical User Interface (GUI). Most designs include Verilog RTL modules, with complex inputs and outputs, which implement a color filter, a pixel processor to calculate the proximity and alignment (centroid) of the target, and the differential drive controller to set the speed of the two separately driven wheels.

Fig. 7 shows a diagram of the Verilog modules used in this POC testing. The Verilog design receives 160 × 120 pixel images with an RGB color depth of 12 bits, stored in an input framebuffer. This input image is filtered based on a color that can be configured by a 3-bit input port. As a result of the color processing, the 8-bit "Centroid" and the 3-bit proximity signals are generated for the differential drive motor control module. In addition, a processed frame buffer is included to facilitate the observation and debugging of the filtering algorithm. A detailed description of these modules is available in [4].

The Motors Control module generates the speed for left and right wheels (in Degrees Per Second, DPS) to direct the robot toward the detected object thanks to the Proximity and Centroid output signals. The values are selected to track the object and achieve the reference proximity value of '1'. The SimMotor connector translates those DPS to linear and rotation speeds, which are then published via the `/cmd_vel` ROS topic back to the Gazebo simulated world where the robot begins to move.

Table 1
Processing times for different camera frame sizes.

Frame size	Processing time (ms)
QQVGA (160 × 120)	41
QVGA (320 × 240)	156
VGA (640 × 480)	680

4.3. Typical execution

Fig. 8 represents a screenshot from the running of this POC, where the GUI of the simulation system can be seen working. The Gazebo world simulation with the simulated robot and human is on the right side. On the left side is the GUI of the Display connector; notice that the red filter is selected but can be interactively modified by enabling/disabling the different color boxes. In the processed output display, the man's red suit can be seen as detected, and its position and distance are displayed with the GUI LEDs. It is observed that the robot moves toward the detected object.

An example of this POC running can be seen in this video[14] (hereafter, Final Video), in which the robot successfully tracks and follows the person moving in the simulated world.

4.4. Simulation performance

These experiments have been performed on a computer running Ubuntu 20.04 LTS with Intel Core i5-13500 13th Gen. using 32 GB of DDR4 3200 MHz RAM, and a AMD Radeon RX 6600 8 GB graphics card. For the described POC and these computer characteristics, the simulation was performed in real-time when the camera's frame size was scaled down to 160 × 120 pixels (QQVGA resolution), achieving a processing time of 41 ms per frame, equivalent to approximately 24 frames per second.

For higher resolutions, the processing time increases significantly, as shown in Table 1, mainly due to the computational demands of the Verilated FPGA simulation. At QVGA resolution (320 × 240 pixels), the processing time increases to 156 ms per frame, allowing only 7 frames per second. For VGA resolution (640 × 480 pixels), the processing time reaches 680 ms per frame, dropping the frame rate to less than 2 frames per second.

[10] https://github.com/JdeRobot/FPGA-robotics/tree/master/blocks
[11] https://github.com/JdeRobot/FPGA-robotics/tree/master/sim_fpga/paper-FPGA_Robotics-sim-FPGA
[12] https://github.com/aws-robotics/aws-robomaker-small-house-world
[13] https://github.com/JdeRobot/FPGA-robotics/tree/master/sim_fpga
[14] Final Video: https://youtu.be/aude7x1EyQs

Fig. 5. Screenshot of a simulated environment in Gazebo, featuring a person dressed in red and a TurtleBot2 robot.

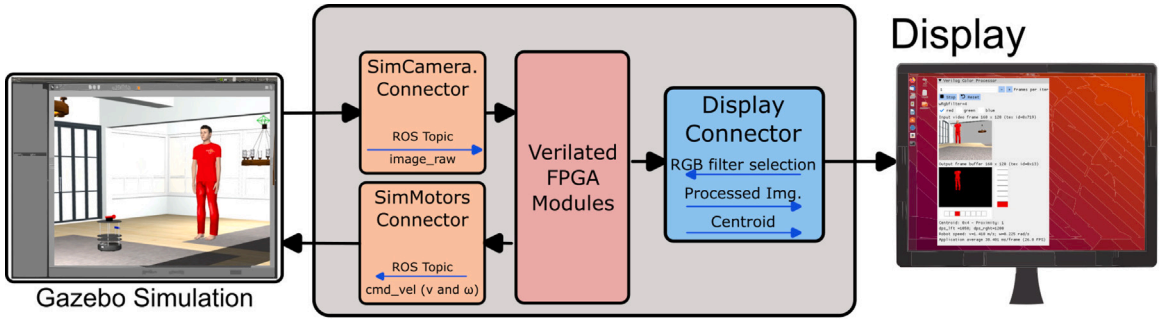


Fig. 6. Simplified diagram outlining the main blocks of the robotics application proof of concept (POC). The system consists of Verilated Modules, a GUI for displaying input and output images, SimCamera and SimMotors connectors, and communication logic with the Gazebo simulated world.

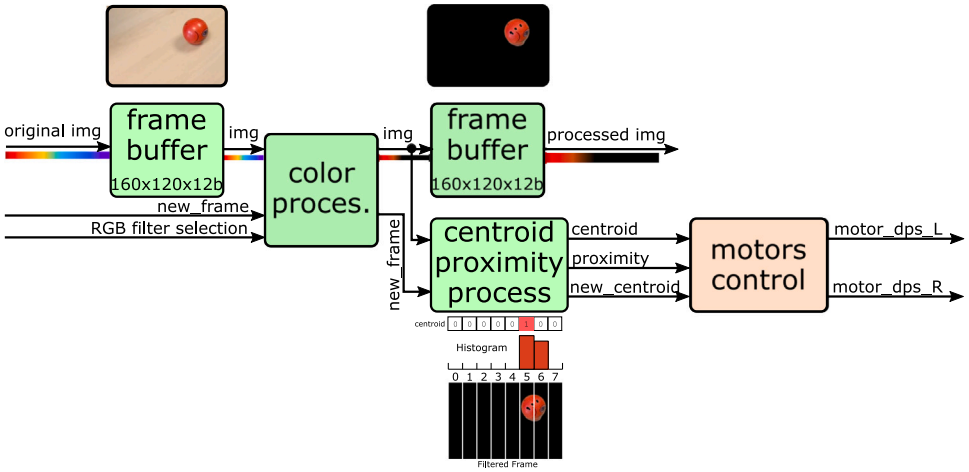


Fig. 7. Simplified Verilog module diagram used in POC testing. These modules process 160 × 120-pixel images with 12-bit RGB color depth. The output signals control the robot's movement towards detected objects.

These measurements include the entire processing pipeline: Verilated FPGA simulation, connectors, and SDL2-based visualization. However, profiling indicates that the main bottleneck is the Verilator simulation, which dominates the processing time.

For larger frame sizes or more complex FPGA processing, the i5-13500 computer running the Verilated code along the Gazebo simulation is not able to reach real-time simulation. For these cases, Gazebo allows controlling the size of the simulation time step in order to run slower than real-time.

4.5. Experimental validation in real robot

The proposed methodology has also been experimentally validated by implementing different vision-based robotic applications in a real robot (GopiGo robot model in this case, also with differential drive motors and a camera). The details of this implementation are extensively described in a previous paper [4] and the Verilog code is available

Fig. 8. Simulation of a robot tracking a person, showing the simulation system GUI alongside the Gazebo world simulation with a robot and a person. The Display connector GUI features a selectable red filter, with the processed output indicating the detected red suit and the robot's movement towards it. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

in the project's repository.[15] As described in the Discussion section, the proposed methodology proved to be very useful for debugging and tuning the robot's behavior.

5. Discussion

The POC detailed in the previous section demonstrates the benefits of including a new verification stage that integrates the FPGA cycle-accurate simulation within the robot behavioral simulation. The proposed approach promotes a development process in which the FPGA-based robotics applications are more completely verified before validating them in the real robot. With the help of Verilator FPGA simulator, Gazebo robotics simulator, ROS, and other FPGA tools, it provides a full development system and the needed execution environments.

The proposed approach brings several advantages to this space: easier debugging, fewer resources needed, faster development times, flexible testing environments, and safer validation with real robots. For instance, during the development of this POC, we were able to refine three fundamental characteristics of the robot's behavior: fine-tuning the color filter, the levels of proximity signal, and the motor control. The correct robot's behavior was shown in the Final Video (see link in Section 4.3). In addition, a video with examples of the debugging process in which the robot did not work properly is available here[16] (hereafter Debug Video).

5.1. Easier debugging

The debugging process benefits significantly from the visualization tools provided by the framework. For example, during the development of the color filter module, visual debugging allowed the quick identification of issues related to shadow gradients and unexpected reflections. These issues were observed in real-time by analyzing the filtered images alongside the original ones. Fig. 9 shows the original behavior of the red color filter before adjusting it to the room conditions. As it can observed from Figs. 9.A and 9.B, the wooden divider at the back is also detected as red. This is not a problem when the target is close, as in Fig. 9.B, because the robot is able to detect the target due to its proximity. However, if the robot misses the target (Fig. 9.D), it will detect the divider as a distant target, trying wrongly to reach it. This behavior can be observed from time 0^m47^s of the Debug Video.

Note that this behavior is very convenient to debug using the proposed methodology because the changes in lightning, the shadows, and the variations due to the orientation are automatically handled by the Gazebo simulator. Moreover, the ability to visualize both the captured and the processed images facilitates the filter refinement and detection of bugs. On the other hand, this type of behavior is difficult to debug using conventional testbenches, not only due to the difficulties in modeling a stream of realistic camera images taking into account the lightning variations but also interpreting the filter's behavior with the simulation waveforms.

Finally, to perform a standalone test of the color filter without the Gazebo robot simulator, the simulation system also allows connecting physical elements to the computer, such as a camera. As an example, Fig. 10 shows the system receiving the inputs from a real camera connected to the computer. This is an important advantage for the specific case of image processing because the color may vary from camera to camera, and the system receives the real image from the camera with the current illumination conditions.

In quantitative terms, debugging the color filter using the framework reduced the time required to fix the issue from an estimated 2 h (using traditional waveforms and testbenches) to just 30 min. This speedup is due to the ability to visually interpret the filter's behavior and adjust it interactively, rather than decoding numerical waveforms manually.

5.2. Reduced resource usage

The integration of the Verilator FPGA simulator with Gazebo reduces the dependency on hardware testbenches, which can be resource-intensive. Table 1 summarizes the processing times for different frame

[15] https://github.com/JdeRobot/FPGA-robotics/tree/master/phys_fpga
[16] Debug Video: https://youtu.be/tW-DOK6nQh4

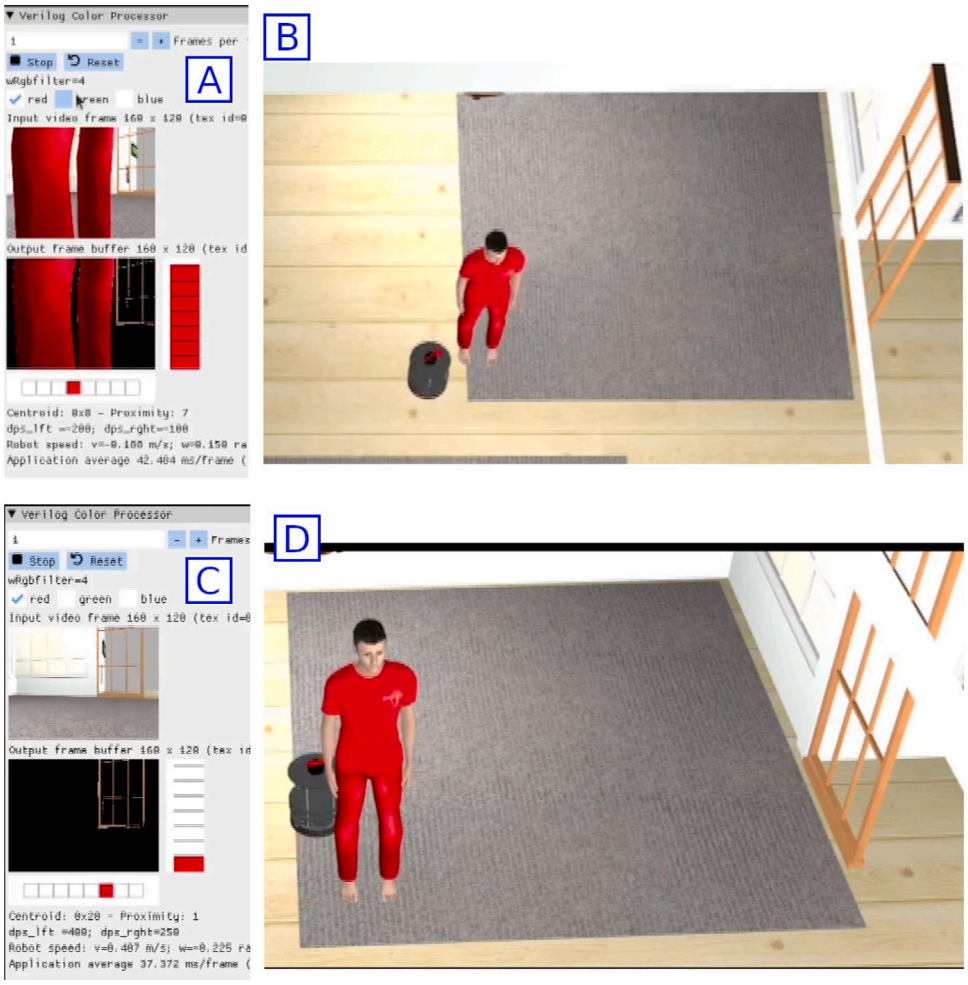


Fig. 9. Screenshots of the robot simulation before tuning the color filter. A and C show the system's GUI with the original captured and the color-filtered images. B and D show the scene including the robot and the target person for A and C, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

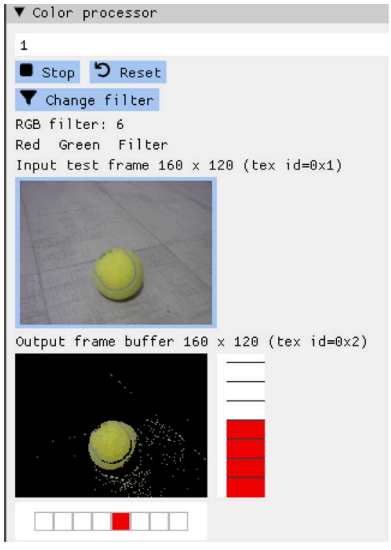


Fig. 10. Screenshot of the GUI to test the yellow color filter with a real camera connected to the computer. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

resolutions in the Verilator simulation, demonstrating that high accuracy in FPGA behavior is achieved without needing physical FPGA boards for initial validation stages. This resource efficiency enables iterative design processes to be conducted on standard desktop computers, bypassing the need for expensive and power-hungry FPGA prototyping hardware during early development stages.

5.3. Faster development times

The ability to seamlessly integrate FPGA simulation with Gazebo and ROS speeds up the development cycle. For example, in the development of the proximity signal processing module, where the proximity signal (see Fig. 7) indicates the closeness to the detected target, in a range from zero to seven. The value of this signal is calculated from the number of pixels that pass through the color filter. However, determining the levels of this signal depends on the target shape, the aspect ratio of the camera, and other characteristics such as the camera's location within the robot and its inclination. Consequently, determining how many red pixels define the desired target distance can be a difficult task to complete theoretically. However, it can be easy to adjust empirically after a few simulations. For example, Fig. 11 shows a previous version in which the value of the proximity signal was too small when the robot was touching the target. In that example, the proximity level was equal to 5, but the robot was trying to get closer, thus colliding with the target. This behavior can be observed from time 0^m23^s of the Debug Video. The entire tuning process, including

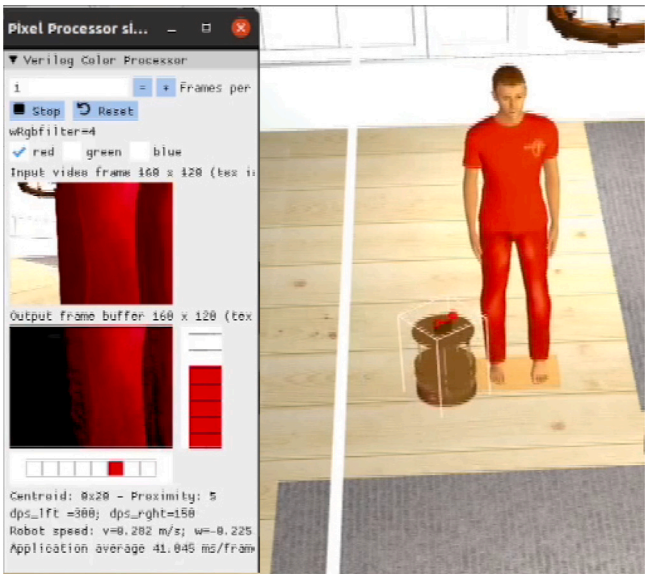


Fig. 11. Screenshot of the robot simulation colliding with the target due to a low value of the Proximity signal.

debugging and testing, took approximately 4 h using the proposed framework. In contrast, performing similar adjustments on physical hardware would have required multiple assembly and testing cycles, estimated to take at least 8 h.

Thanks to the system's comprehensive information, we were able to adjust the thresholds for the proximity signal levels easily. On the one hand, the Gazebo simulation shows the scene including the robot and the target, making it straightforward to determine how close the target is to the robot and the occurrence of any collision. On the other hand, the system's GUI provides the internal FPGA processing information, such as the filtered image, the proximity value, and each of the motor's speed and direction.

In addition, our simulation scheme facilitated finding common bugs in the FPGA code itself. For example, in an initial version of the design, it was easy to observe that the centroid signal was not centered with the filtered image. The bug was caused by having a signal with a lower bit range than needed, which can be considered a typical bug when describing digital circuits.

5.4. Safer validation with real robots

The proposed methodology minimizes risks associated with testing FPGA designs on real robots. For example, an unstable PID control module that caused erratic motor behavior was debugged and stabilized in simulation before being deployed to hardware. The ability to test dangerous scenarios in a safe, simulated environment is particularly advantageous when working with expensive or delicate robotic systems.

The Motors Control module (see Fig. 7) implements a proportional control for the differential wheeled robot. This module receives the Proximity and Centroid signals, providing the angular speed for each of the two driving wheels, left and right. Adjusting the control constants (such as K_p) can be done theoretically, by trial and error, or using a combination of both. The visual simulation provides a safe environment for testing different control strategies and fine-tuning the constants. The ability to observe the robot trajectory, its oscillations, and instabilities is a clear advantage compared with lower-level simulations, in which interactive inputs are cumbersome to generate and the robot trajectory is difficult to infer.

In addition, the simulation environment allows changing the physical conditions, such as the dimensions and weight of the robot, or the slope and the kind of surface where the robot is moving along. The proposed methodology provides an agile way to adjust the control to those new situations.

Fig. 12 shows a simulation in which the control was so unstable that it caused the robot to fall. This behavior can be observed from time 2^m32^s of the Debug Video.

6. Conclusions

This article proposes a novel methodology for developing FPGA-based robotics applications that combines simulations at two abstraction levels: a higher level for the robot's behavior and a lower level for the FPGA cycle-accurate simulation. Our proposal offers exceptional advantages because it enables the verification of the robot's performance while being able to graphically observe the internal processing of the FPGA signals.

The second contribution is the development of an open-source combined simulation scheme that supports the proposed methodology. It includes Verilator for the functional simulation of Verilog source code, ROS and Gazebo for the robot simulation, and several *connectors* that tie the Verilog code to robot sensors and actuators through ROS topics.

The methodology has been experimentally validated successfully developing the vision-based "Follow Person" application with a commercial TurtleBot2 robot, as POC. The proposed methodology has allowed a fast fine-tuning of the Verilog code for the color filter and the motion controller. Quantitative results for simulation performance were included in this study, showing that the combined simulation allows real-time operation with smaller frame sizes (e.g., QQVGA at 41 ms per frame) and manageable processing times for higher resolutions. The same application was also transferred to a real robot in a recent work [4]. The development of several FPGA-based modules for robots has been portrayed. These applications, tests, and documentation have contributed to and will continue to help with the work being carried out in the international open-source reconfigurable computing community.

As this research shows, transitioning part of the development process from physical hardware to simulated environments can provide significant advantages like reduced costs and faster and more versatile designing, prototyping, and testing processes. These benefits of the proposed methodology are already evident from the POC and modules discussed. Being able to debug FPGA applications that use live image data without having to create realistic testbenches significantly accelerates development. For instance, the debugging of FPGA applications

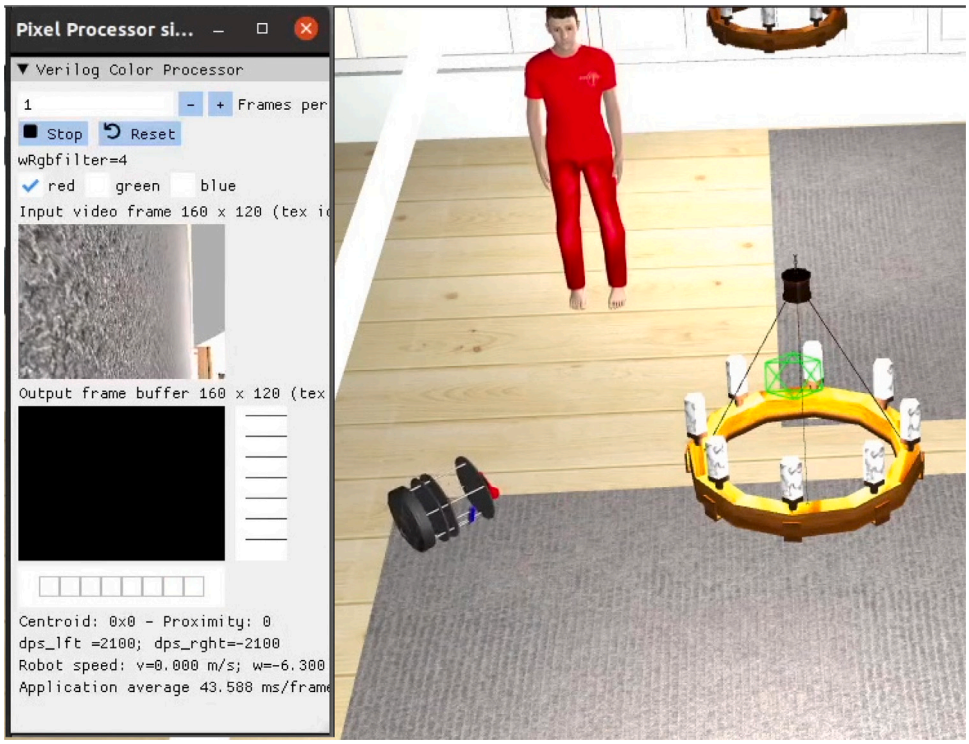


Fig. 12. Screenshot of the robot simulation failure when motor control was too unstable.

using live image data was accelerated by at least 50%, as demonstrated by the quantitative case studies. Not needing to synthesize and upload the modules to the actual board dramatically helps with the application testing process, facilitating the verification that everything works correctly.

The work presented in this document opens new research paths and establishes the foundations for developing more complex FPGA Modules for robots. The created simulation scheme and tools contribute to a living open-source ecosystem where hardware and software are constantly improved.

Regarding extensibility, we acknowledge that some components, such as the connectors, are currently manually written. However, the system's architecture has been designed to facilitate modularity and future automation of connector generation.

As main future lines, the development of new connectors, such as one for point cloud sensors (e.g., LIDAR), is an interesting extension of this work. We are also working on introducing a basic trained neural network in the simulated FPGA, implemented in Verilog, for instance for the vision-based "Follow Road" robotics application. A performance comparison between the same robotics applications deployed on FPGAs and ordinary microprocessors is planned. In addition, a deep performance analysis when scaling to more complex Verilog robotics applications.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was partially funded by Spanish Agencia Estatal de Investigación through the project (GAIA) project "Gestión integral para la prevención, extinción *y* reforestación debido a incendios forestales", Proyectos de I+D en líneas estratégicas en colaboración entre organismos de investigación *y* difusión de conocimientos TRANSMISIONES 2023. Ref PLEC2023-010303 (2024–2026), and the (UNIBOTICS-GAM) project "Plataforma web educativa abierta para la programación de robots en ingeniería", Proyectos de Transición Ecológica *y* Transición Digital 2021. Ref TED2021-132632B-I00. (2022–2024)

Data availability

No data was used for the research described in the article.

References

[1] V. Mayoral-Vilches, G. Corradi, Adaptive computing in robotics, leveraging ROS 2 to enable software-defined hardware for FPGAs, 2021, arXiv preprint arXiv:2109.03276.
[2] A. Podlubne, D. Göhringer, Modeling FPGA-based architectures for robotics, in: 2022 International Conference on Field-Programmable Technology, ICFPT, IEEE, 2022, pp. 1–4.
[3] C. Lienen, M. Platzner, Design of distributed reconfigurable robotics systems with ReconROS, ACM Trans. Reconfigurable Technol. Syst. 15 (3) (2022).
[4] F. Machado, R. Nieto, J. Fernández-Conde, D. Lobato, J.M. Cañas, Vision-based robotics using open FPGAs, Microprocess. Microsyst. (2023) 104974.
[5] C. Wolf, Project IceStorm - lattice ice40 FPGAs bitstream documentation (reverse engineered), 2015, https://github.com/YosysHQ/icestorm.
[6] E.D. Sozzo, D. Conficconi, A. Zeni, M. Salaris, D. Sciuto, M.D. Santambrogio, Pushing the level of abstraction of digital system design: A survey on how to program FPGAs, ACM Comput. Surv. 55 (5) (2022) 1–48.
[7] J. Bergeron, Writing testbenches: functional verification of HDL models, Springer Science & Business Media, 2012.
[8] H.D. Foster, 2018 FPGA functional verification trends, in: 2018 19th International Workshop on Microprocessor and SOC Test and Verification (MTV), 2018, pp. 40–45, ISSN: 2332-5674.
[9] T.S. Tan, B.A. Rosdi, Verilog HDL simulator technology: a survey, J. Electron. Test. 30 (2014) 255–269.
[10] D.-L. Lin, H. Ren, Y. Zhang, B. Khailany, T.-W. Huang, From RTL to CUDA: A GPU acceleration flow for RTL simulation with batch stimulus, in: Proceedings of the 51st International Conference on Parallel Processing, 2022, pp. 1–12.
[11] S. Beamer, A case for accelerating software RTL simulation, IEEE Micro 40 (4) (2020) 112–119.
[12] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng, ROS: an open-source robot operating system, in: ICRA Workshop on Open Source Software, Vol. 3, No. 3.2, Kobe, Japan, 2009, p. 5.

[13] M. Quigley, B. Gerkey, W.D. Smart, Programming Robots with ROS: a practical introduction to the Robot Operating System, " O'Reilly Media, Inc.", 2015.
[14] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot operating system 2: Design, architecture, and uses in the wild, Sci. Robot. 7 (66) (2022) eabm6074.
[15] A. Farley, J. Wang, J.A. Marshall, How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, gazebo, MORSE and webots with a focus on accuracy of motion, Simul. Model. Pr. Theory 120 (2022) 102629.
[16] Z. Wan, B. Yu, T.Y. Li, J. Tang, Y. Zhu, Y. Wang, A. Raychowdhury, S. Liu, A survey of FPGA-based robotic computing, IEEE Circuits Syst. Mag. 21 (2) (2021) 48–74.
[17] A. Podlubne, D. Göhringer, A survey on adaptive computing in robotics: Modelling, methods and applications, IEEE Access (2023).
[18] Z. Wan, A. Lele, B. Yu, S. Liu, Y. Wang, V.J. Reddi, C. Hao, A. Raychowdhury, Robotic computing on FPGAs: Current progress, research challenges, and opportunities, in: 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems, AICAS, IEEE, 2022, pp. 291–295.
[19] S. Liu, Z. Wan, B. Yu, Y. Wang, FPGA technologies, in: Robotic Computing on FPGAs, Springer, 2022, pp. 11–29.
[20] A. Podlubne, J. Mey, R. Schöne, U. Aßmann, D. Göhringer, Model-based approach for automatic generation of hardware architectures for robotics, IEEE Access 9 (2021) 140921–140937.
[21] X. Shi, L. Cao, D. Wang, L. Liu, G. You, S. Liu, C. Wang, HERO: Accelerating autonomous robotic tasks with FPGA, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2018, pp. 7766–7772.
[22] M. Bouhali, F. Shamani, Z.E. Dahmane, A. Belaidi, J. Nurmi, FPGA applications in unmanned aerial vehicles-a review, in: International Symposium on Applied Reconfigurable Computing, Springer, 2017, pp. 217–228.
[23] B. Cain, Z. Merchant, I. Avendano, D. Richmond, R. Kastner, PynqCopter-an open-source FPGA overlay for UAVs, in: 2018 IEEE International Conference on Big Data (Big Data), IEEE, 2018, pp. 2491–2498.
[24] F.F. Nyboe, N.H. Malle, E. Ebeid, MPSoC4Drones: An open framework for ROS2, PX4, and FPGA integration, in: 2022 International Conference on Unmanned Aircraft Systems, ICUAS, IEEE, 2022, pp. 1246–1255.
[25] M.-A. Martínez-Prado, J. Rodríguez-Reséndiz, R.-A. Gómez-Loenzo, G. Herrera-Ruiz, L.-A. Franco-Gasca, An FPGA-based open architecture industrial robot controller, IEEE Access 6 (2018) 13407–13417.
[26] C. Lienen, M. Brede, D. Karger, K. Koch, D. Logan, J. Mazur, A.P. Nowosad, A. Schnelle, M. Waizy, M. Platzner, AutonomROS: A ReconROS-based autonomous driving unit, 2023.
[27] J. Caro, A. Barrientos, E. Mayas, Hybrid bio-inspired architecture for walking robots through central patter generators using open source FPGAs, in: Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on, IEEE, 2018.
[28] K. Yamashina, T. Ohkawa, K. Ootsu, T. Yokota, Proposal of ROS-compliant FPGA component for low-power robotic systems, 2015, CoRR, abs/1508.07123.
[29] C. Lienen, M. Platzner, B. Rinner, ReconROS: Flexible hardware acceleration for ROS2 applications, in: 2020 International Conference on Field-Programmable Technology, ICFPT, 2020, pp. 268–276.
[30] C. Lienen, M. Platzner, Reconros executor: Event-driven programming of FPGA-accelerated ROS 2 applications, 2022, arXiv preprint arXiv:2201.07454.
[31] A. Podlubne, D. Göhringer, FPGA-ROS: Methodology to augment the robot operating system with FPGA designs, in: 2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig), IEEE, 2019, pp. 1–5.
[32] Y. Sugata, T. Ohkawa, K. Ootsu, T. Yokota, Acceleration of publish/subscribe messaging in ROS-compliant fpga component, in: Proceedings of the 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, 2017, pp. 1–6.
[33] T. Ohkawa, Y. Sugata, H. Watanabe, N. Ogura, K. Ootsu, T. Yokota, High level synthesis of ROS protocol interpretation and communication circuit for FPGA, in: 2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering (RoSE), IEEE, 2019, pp. 33–36.
[34] C. Lienen, S.H. Middeke, M. Platzner, fpgaDDS: An intra-FPGA data distribution service for ROS 2 robotics applications, 2023.
[35] J.P. Queralta, F. Yuhong, L. Salomaa, L. Qingqing, T.N. Gia, Z. Zou, H. Tenhunen, T. Westerlund, FPGA-based architecture for a low-cost 3D lidar design and implementation from multiple rotating 2D lidars with ROS, in: 2019 IEEE Sensors, IEEE, 2019, pp. 1–4.
[36] J. Vourvoulakis, J. Kalomiros, J. Lygouras, FPGA-based architecture of a real-time sift matcher and RANSAC algorithm for robotic vision applications, Multimedia Tools Appl. 77 (8) (2018) 9393–9415.
[37] R. Miyagi, N. Takagi, S. Kinoshista, M. Oda, H. Takase, Zytlebot: FPGA integrated ROS-based autonomous mobile robot, in: 2021 International Conference on Field-Programmable Technology, ICFPT, IEEE, 2021, pp. 1–4.
[38] M. Eisoldt, S. Hinderink, M. Tassemeier, M. Flottmann, J. Vana, T. Wiemann, J. Gaal, M. Rothmann, M. Porrmann, Reconfros: Running ROS on reconfigurable SOCs, in: Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings, 2021, pp. 16–21.
[39] M. Eisoldt, M. Flottmann, J. Gaal, S. Hinderink, J. Vana, M. Tassemeier, M. Rothmann, T. Wiemann, M. Porrmann, ReconfROS: An approach for accelerating ROS nodes on reconfigurable socs, Microprocess. Microsyst. 94 (2022) 104655.
[40] N. Malle, E. Ebeid, Open-source educational platform for FPGA accelerated AI in robotics, in: 2022 8th International Conference on Mechatronics and Robotics Engineering, ICMRE, IEEE, 2022, pp. 112–115.
[41] R. Liu, J. Yang, Y. Chen, W. Zhao, Eslam: An energy-efficient accelerator for real-time orb-slam on FPGA platform, in: Proceedings of the 56th Annual Design Automation Conference 2019, 2019, pp. 1–6.
[42] K. Boikos, C.-S. Bouganis, A scalable FPGA-based architecture for depth estimation in SLAM, in: International Symposium on Applied Reconfigurable Computing, Springer, 2019, pp. 181–196.
[43] V.H. Schulz, F.G. Bombardelli, E. Todt, A harris corner detector implementation in SoC-FPGA for visual SLAM, in: Robotics, Springer, 2016, pp. 57–71.
[44] J. Rodriguez-Araujo, J.J. Rodriguez-Andina, J. Farina, M.-Y. Chow, Field-programmable system-on-chip for localization of UGVs in an indoor ispace, IEEE Trans. Ind. Informat. 10 (2) (2013) 1033–1043.
[45] F.S. Alkhafaji, W.Z. Hasan, M. Isa, N. Sulaiman, Robotic controller: ASIC versus FPGA—A review, J. Comput. Theor. Nanosci. 15 (1) (2018) 1–25.
[46] J. Pérez, A. Alabdo, J. Pomares, G.J. García, F. Torres, FPGA-based visual control system using dynamic perceptibility, Robot. Comput.-Integr. Manuf. 41 (2016) 13–22.
[47] A. Alabdo, J. Pérez, G.J. Garcia, J. Pomares, F. Torres, FPGA-based architecture for direct visual control robotic systems, Mechatronics 39 (2016) 204–216.
[48] G. Quintal, E.N. Sanchez, A.Y. Alanis, N.G. Arana-Daniel, Real-time FPGA decentralized inverse optimal neural control for a shrimp robot, in: 2015 10th System of Systems Engineering Conference, SoSE, IEEE, 2015, pp. 250–255.
[49] R. Chand, R.P. Chand, M. Assaf, P.R. Naicker, S.V. Narayan, A.F. Hussain, Embedded FPGA-based motion planning and control of a dual-arm car-like robot, in: 2022 IEEE 7th Southern Power Electronics Conference, SPEC, IEEE, 2022, pp. 1–6.
[50] S. Murray, W. Floyd-Jones, Y. Qi, D.J. Sorin, G. Konidaris, Robot motion planning on a chip., in: Robotics: Science and Systems, 2016.
[51] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, D.J. Sorin, The microarchitecture of a real-time robot motion planning accelerator, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, IEEE, 2016, pp. 1–12.
[52] Y. Zhou, X. Jin, T. Wang, FPGA implementation of a* algorithm for real-time path planning, Int. J. Reconfigurable Comput. 2020 (2020) 1–11.
[53] E. Moréac, E.M. Abdali, F. Berry, D. Heller, J.-P. Diguet, Hardware-in-the-loop simulation with dynamic partial FPGA reconfiguration applied to computer vision in ROS-based UAV, in: 2020 International Workshop on Rapid System Prototyping, RSP, 2020, pp. 1–7.
[54] C. Lienen, M. Platzner, Task mapping for hardware-accelerated robotics applications using reconros, in: 2022 Sixth IEEE International Conference on Robotic Computing, IRC, 2022, pp. 148–155.

Rubén Nieto received his Ph.D. degree (with honours) in Electronics: Advanced Electronic Systems and Intelligent Systems from the University of Alcalá (UAH), Spain, in 2020. Currently, he is working as Assistant Professor at Electronics Technology Department, Rey Juan Carlos University. His areas of research interests include Mobile Robots, Multisensor Integration, Multiprocessor System-on-Chip, and Digital and Embedded Systems.

Felipe Machado received Industrial Engineering degree in 1998, and a Ph.D. in 2008, both from the Universidad Politécnica de Madrid. He has been assistant professor at Universidad Rey Juan Carlos for more than ten years. Currently he is a researcher with the Institute for Applied Microelectronics at Universidad de Las Palmas de Gran Canaria. His research interests include video coding systems, reconfigurable architectures, robotics, and development of open-source scientific equipment.

Jesús Fernández-Conde received the M.S. degree in telecommunications engineering from the Universidad Politécnica de Madrid, in 1994, and the Ph.D. degree in computer science from the Universidad Complutense de Madrid, in 2011. He is currently an Associate Professor with the Department of Signal Theory and Communications, Universidad Rey Juan Carlos, Spain. His research interests include real-time systems, artificial intelligence applications, and robotics.

David Lobato received Computer Engineering degree in 2008 and Master of Science degree in 2010, both from Universidad Rey Juan Carlos de Madrid. Currently he is a Software Engineer at Turbine Kreuzberg Portugal developing IoT systems. He also participates with Jderobot NGO in robotics related projects. His interest includes robotics, computer vision and reconfigurable computing.

José M. Cañas received the M.S. and Ph.D. degrees in telecommunications engineering from the Universidad Politécnica de Madrid. He has done research in robotics at Carnegie Mellon University, USA, the Georgia Institute of Technology, USA, the Instituto Nacional de Astrofísica, Óptica y Electrónica, México, and the Instituto de Automática Industrial (CSIC). His research interests include robotics, computer vision, and the education of those disciplines.