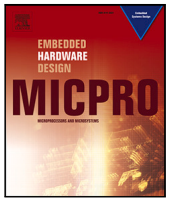
Microprocessors and Microsystems 103 (2023) 104974

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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



Vision-based robotics using open FPGAs

Felipe Machado [a,b], Rubén Nieto [a], 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:
FPGA
Computer vision
Robotics
Open-source

ABSTRACT

Robotics increasingly provides practical applications for society, such as manufacturing, autonomous driving, robot vacuum cleaners, robots in logistics, drones for inspection, etc. Typical requirements in this field are fast response time, low power consumption, parallelism, and flexibility. According to these features, FPGAs are a suitable computing substrate for robots. A few vendors have dominated the FPGA market with their proprietary tools and hardware devices, resulting in fragmented ecosystems with few standards and little interoperation. New and complete open toolchains for FPGAs are emerging from the open-source community. This article presents an open-source library of Verilog modules useful for vision-based robots, including reusable image processing blocks for perception and reactive control blocks. This library has been developed using open tools, but its Verilog modules are fully compatible with any proprietary toolchain. In addition, three applications with a real robot and open FPGAs have been developed for experimental validation using this library. In the last application, the mobile robot successfully follows a colored object using two low-cost cameras (to increase the robot's field of view) and includes a third camera on top of a servo-driven turret for tracking a second independent object while following the first one in parallel. Resource consumption of all applications has been measured and compared with state-of-the-art proprietary toolchains, revealing that reconfigurable computing with open FPGAs using open tools is now an attractive alternative to designing and creating intelligent vision-based robotic applications using vendor-dependent proprietary tools and FPGAs.

1. Introduction

Robotics is an exciting engineering field with recent massive applications beyond the classic automotive and integrated circuit factories. Logistics (such as Amazon robots at warehouses), food packaging, autonomous driving cars, drones for inspection, and home vacuum cleaners are just a few examples. Robots have departed from research labs and are increasingly entering people's daily life environments.

Real-world robotics applications commonly require reliability and real-time operation. Robot behaviors must be robust and agile, even when using cameras as the main sensor input. Low power consumption is also a requirement in most cases, as in drones or mobile robots.

Robots are composed of hardware and software. Sensors, actuators, and a computation substrate are the main hardware components. Sensors provide information about the surroundings, such as laser scanners, LIDAR, cameras, battery sensors, Inertial Measurement Units, encoders, etc. Actuators allow the robot to perform actions, including the robot's physical movement. Electrical motors are the most ubiquitous ones. Typically there are four computing substrates: CPUs, GPUs, FPGAs, and ASICs. The most widely used are microprocessors (CPUs) or microcontrollers, which execute robotics software. Reconfigurable computing, commonly implemented using FPGAs, is also an attractive alternative, as their low price, fast execution speed, power efficiency, and reconfigurability are clearly beneficial in robotics.

FPGAs have been used in many application fields [1], such as digital control, communication interfaces, networking, computer security, cryptography techniques, machine learning, digital signal processing, image and video processing, big data, and computer algorithms. Two leading vendors dominate the FPGA market: AMD-Xilinx and Intel-Altera, with more than 85% of the share. They provide FPGA circuits and proprietary development tools (such as Vivado and Vitis from AMD-Xilinx or Quartus-II from Intel-Altera). This closed market has resulted in a fragmented ecosystem and low interoperation.

In the last few years, various open tools for development with FPGAs have appeared [2–4], some of them based on reverse engineering of the devices from the most extended providers [5,6]. New projects have recently emerged to group these open tools in a single toolchain, such as Apio [7], Icestudio [8], the OSS CAD Suite [9] and F4PGA [10]. The latter is a representative case, in which the F4PGA Workgroup has been

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

https://doi.org/10.1016/j.micpro.2023.104974
Received 4 September 2022; Received in revised form 1 July 2023; Accepted 13 November 2023
Available online 14 November 2023

established by leading FPGA vendors, industrial FPGA users as well as several prestigious universities to accelerate the adoption of open source FPGA tools.

The intersection of robotics and reconfigurable computing is an emerging field. FPGAs are increasingly used to accelerate parts of standard robotics computations such as perception, localization, motion planning, or control. For instance, inside the ROS ecosystem [11] (de facto standard in robot programming), a Working Group has been recently created on Hardware Acceleration [12]. Its mission is to drive the creation, maintenance, and testing of acceleration kernels on top of open standards (C++ and OpenCL) for optimized ROS 2 and Gazebo interactions over different computation substrates (including FPGAs, GPUs, and ASICs).

This paper presents `FPGA-Robotics`[13], an open library of Verilog modules available for vision-based robotics. Some image processing, sensor, and motor control modules have been developed and are ready to use. They have been assembled in several robotic and computer vision applications (for instance, a robot following two colored objects simultaneously by managing three cameras in parallel) using an open toolchain for FPGAs based on Yosys [2] and nextpnr [14] tooling.

Open tools are cross-platform and therefore release developers from being tied to a single vendor. Their developments may be easily migrated from one hardware to another, even from different manufacturers. Besides, the Verilog modules of the `FPGA-Robotics` library are fully compatible with different proprietary FPGA toolchains of the prevalent FPGA vendors (Lattice/Xilinx(AMD)/Intel-Altera).

The resource consumption of the three developed applications has been compared for the Lattice, Xilinx(AMD), Intel-Altera, and open toolchain, using different FPGA boards. The performance of the open toolchain is similar to that of the proprietary toolchain on the same hardware, showing better results concerning the number of RAM memory blocks and performing worse regarding logic blocks. As the library's purpose is to build vision-based robotic applications, this behavior is beneficial in most cases.

Experimental validation demonstrates that reconfigurable computing *with recent open FPGA tools* is a cost-effective and reliable approach to developing reactive robot behaviors involving computer vision. The present work is a step forward from previous ones implementing reactive robot applications using open FPGAs based on simple robotic sensors, such as inertial [15], or IR and sonars [16].

The remainder of this paper is organized as follows. Section 2 reviews related research works in the literature combining FPGAs and robotics. Section 3 describes the open toolchain used and the developed Verilog modules, composing the proposed open FPGA developer framework for vision-based robotics applications. Section 4 presents the experimental validation with three vision-based robot reactive behaviors implemented utilizing the proposed framework, including a quantitative comparison with state-of-the-art proprietary toolchains. Finally, the main conclusions of this work are summarized in Section 5.

2. Related work

Robotics applications typically comprise two stacks connected to the robot sensors and actuators: perception and decision-making. The perception stack includes mapping, localization, object detection, and tracking (among other tasks). The decision stack includes feedback control, obstacle avoidance, and planning. The operation of sensors and actuators requires the corresponding drivers. Those are the classic robotics workloads, and FPGAs have been explored for their accelerated execution and power efficiency in all of them. In this section, several illustrative examples will be reviewed. A deeper and broader survey can be found at Wan et al. [17].

FPGAs have been widely used for speeding up image processing inside robot visual perception. Alabdo et al. [18] describe a complete visual pipeline on FPGA, including thresholding, erosion, blob detection, and center calculation. FPGAs have also been used for more elaborate image processing, such as Harris corner detector [19], and extraction and matching of Scale-Invariant Feature Transform (SIFT) keypoints [20]. In recent years, Deep Learning has greatly improved the robustness of computer vision and made significant progress in solving robot perception problems. FPGAs are also being explored to accelerate Deep Learning based image processing. [21,22] are two good overviews of neural network inference accelerators based on FPGA and the main techniques used. The Xilinx Zynq platform has been used to speed up the inference of Convolutional Neural Networks (CNNs), as in ZynqNet Embedded CNN [23], which was designed for image classification on ImageNet. It has also been used in [24], which describes an open-source framework for designing and implementing a simple neural network targeting edge computing platforms, and in the optimization of Recurrent Neural Networks [25]. Zhang et al. [26] describe an OpenCL-based FPGA accelerator that optimizes CNN classifier kernels. It was tested on an Altera Arria 10 GX1150 board. A recent work [27] explores the CNN kernel partition technique to reduce the repeated access to the input feature maps and the kernels, speeding up the inference time on FPGAs for real-time object recognition applications.

FPGAs have also been used for Simultaneous Localization and Mapping (SLAM), a key capability for mobile robots. Boikos et al. [28] describe an FPGA accelerator architecture for depth estimation in SLAM algorithms achieving a rate of more than 60 mapped fps, similar performance to a high-end desktop CPU with an order of magnitude improved power consumption. Liu et al. [29] proposed eSLAM, an FPGA energy-efficient architecture for the well-known real-time ORB-SLAM algorithm, by accelerating the most time-consuming visual feature extraction and matching stages. They implemented it in the Xilinx Zynq XCZ7045 SoC, achieving a $\times 3$ speed up and a decrease of 1/80 in power consumption.

Performance has also been improved with FPGAs in robot navigation and path planning algorithms. For instance, Murray et al. [30] construct robot-specific circuitry for motion planning, capable of generating motion plans approximately three orders of magnitude faster than traditional methods. Building a probabilistic roadmap is a common approach for motion planning problems. Their proposal makes collision detection circuits for the roadmap edges, which run entirely in parallel to perform the path search. A second example [31] implements a customized Genetic Algorithm for a mobile robot's path planning. A Xilinx FPGA device and a Pioneer 3DX platform were used in this work.

Alkhafaji et al. [32] review several relevant FPGA works in robot control. For control in industrial robots, [33] is an illustrative example. The authors developed an FPGA-based motion control system employing an open architecture and vendor-independent control system. It was tested on a Fanuc S420F using Xilinx FPGAs. Another example with industrial robots is described in [34] for Mitsubishi PA10, incorporating a Xilinx board. A camera served as the primary sensor for servoing control. In addition, Sharma et al. [35] compared several flight control approaches in small Unmanned Aerial Vehicles. FPGA/DSP-based solutions are the best in this domain, as they run with low power, fast response, and less volume and weight. An appealing example is PynqCopter [36], an open-source control system implemented on an FPGA-based board (Xilinx PYNQ-Z1) for a hexacopter. They used High-Level Synthesis tools.

Open FPGA tools such as IceStudio [8] and Apio [7], and the open iCE40 FPGA from Lattice have been used to design robot controllers. Adopting this open FPGA approach, Cañas et al. [16] developed two case-based reactive controllers for the classic FollowLine and ObstacleAvoidance applications with a wheeled robot. Infrared and sonar sensors were employed. Caro et al. [37] control a hexapod robot with an approach inspired by the animal nervous system. It implements the binomial Brain-Peripheral Nervous System (CNS-PNS), combining microprocessors for the high-level control and FPGAs for the low-level control. Central Pattern Generator signals coordinate the motion of all the legs for robot walking.

In real systems, robot computing is typically distributed among FPGAs, general-purpose CPUs, or even Graphics Processing Units (GPUs) in a heterogeneous Hardware–Software co-design. A relevant work showing this is [38], which combines regular software on an Intel Core i5 CPU with FPGA accelerators for several tasks: SLAM, motion planning, and convolutional neural network inference. The OpenCL framework was used for programming and executing programs across heterogeneous platforms. Utilizing FPGA acceleration, the SLAM and motion planning tasks were performed 2-4 times faster than the fine-tuned software implementation.

In recent years, several works have appeared combining the de facto standard in robotics software, Robot Operating System (ROS) [11] with FPGAs. In ROS, a typical robot application is an orchestra of several concurrent nodes, possibly running on different processors, and they interoperate by exchanging messages in the shape of ROS topics/services. The primary approach is to implement some specific ROS nodes inside FPGAs for accelerated execution [39,40], which requires the implementation of the communication protocols of ROS in the FPGA, besides the robotic task itself for those nodes. Another approach is to implement RISC-V processors on FPGAs to run ROS nodes inside them [41].

3. Open framework for reconfigurable computing in robotics

In this section, the framework used in this work will be described. The framework is divided into two parts: first, the open toolchain used for the development of the hardware designs; and second, the proposed open library of hardware designs created to be reused in many different robot applications. This library is available in the FPGA-Robotics repository and it has been implemented using Verilog.

3.1. Open toolchain for FPGAs

The development process in reconfigurable computing includes several steps [42–45] and tools. The hardware may be designed at varying abstraction levels, usually gate level, register-transfer level (RTL), or algorithmic level. Generally, when it comes to developing FPGA designs, the proprietary software of the particular device is employed because the binary format to configure the FPGA is not disclosed by the manufacturer. However, thanks to the aforementioned open FPGA reverse-engineering projects, some FPGA binary formats are available, enabling the generation of the bitstream files to configure the FPGA and thus, the development of complete open-source FPGA toolchains.

Next, we will summarize the general FPGA development process, mentioning the specific open-source tools and methods used in this work:

1. The digital circuit structure and behavior are formally described in a Hardware Description Language (HDL), such as Verilog, VHDL, or recent ones like SpinalHDL. These languages allow for automated analysis and simulation, and their corresponding text files can be seen as the *source code* of a particular application in reconfigurable computing. We have used Verilog for the proposed library and all the designs that will be detailed in the following sections.
2. A logic synthesis tool *synthesizes* the HDL file into a *netlist*. The netlist is a specification of the basic electronic components and their interconnections. We have used the open-source synthesis tool Yosys [2,46] to synthesize our Verilog designs.
3. The *place-and-route* stage maps the basic components of the previously generated netlist to the physical resources of the target FPGA and then decides how to interconnect all the placed components. As a result, the *bitstream* is generated, describing the configuration to be loaded into the FPGA device. nextpnr [14] has been used as an open-source place-and-route tool, obtaining the specific FPGA device information from reverse-engineering projects such as Project Icestorm [47] and Project Trellis [48].
4. Finally, once the bitstream is generated, it can be loaded into the FPGA. There are various open-source tools for this purpose; we have used OpenFPGALoader [49].

If every step has been performed correctly, once the bitstream has been loaded, the FPGA will start functioning as described by the HDL. The process just described involves the use of a set of different tools. Nevertheless, various projects aim to ease the workflow by grouping these tools into a single toolchain. For example, the project Apio [7] is a Python-based tool that integrates all of them in a single command-line interface. For those preferring a graphical interface, Icestudio [8] provides a combination of Verilog and visual schematics for FPGA programming. In addition, the OSS-CAD-Suite provides all these individual tools through a single installation. The F4PGA toolchain [10] is another powerful and illustrative example. The main purpose of F4PGA is to provide an open cross-platform Verilog-bitstream tool for all FPGAs available on the market.

3.2. Open verilog library for vision-based robotics

We have created `FPGA-Robotics`, a library of Verilog modules easy to reuse and integrate into robotics applications. This library is open-source and expandable with new blocks or modules from the developer community.

Among the modules that have been created are a camera driver, a VGA display driver, a driver for the GoPiGo robot [50], and other blocks related to image processing, such as a color filter. Details of some of the modules are listed below. Each module is explained following the flow of image processing, starting with the image acquisition from the camera, continuing with the image processing, and ending with the image visualization or its application to the robotic platform.

All the modules are available in the repository [13] and can be used in any other application. For example, the color processing module can be used with a different camera, adjusting the size of the input image; and the VGA interface module can be used to display any image stored in memory.

3.2.1. Ov7670 camera modules

The ov7670 is a 3 V low-cost VGA camera that provides a maximum resolution of 640×480 pixels in different RGB configurations (RGB565/555/444) and 8-bit YUV 4:2:2 among others. The camera is configured through Serial Camera Control Bus (SCCB), which is an I^2C interface. The camera has more than a hundred registers, which allow configuring several parameters such as AEC (Automatic Exposure Control), AGC (Automatic Gain Control), AWB (Automatic White Balance), as well as color saturation, hue, gamma, among many others. It should be noted that this configuration is only performed once when the device is started. There are two modules developed for the ov7670 camera:

- Configuration module: It comprises two blocks, one to perform the SCCB communication protocol and the other to configure the camera registers.
- Image capture module: The ov7670 camera sends the image in 8-bit synchronous parallel data. For this purpose, the image capture module must provide a clock signal to the camera. In our case, a frequency of 25 MHz has been selected to be able to provide 30 fps. The image acquisition protocol is similar to the VGA protocol. The captured image is stored in a dual port memory (frame buffer) to be able to read and write independently. Most FPGAs have this type of memory, called Block RAM (BRAM). The image capture module writes the originally captured image using the first port, whereas the second port is used to read the stored image to be processed. To reduce the BRAM usage, the image resolution can be decreased.

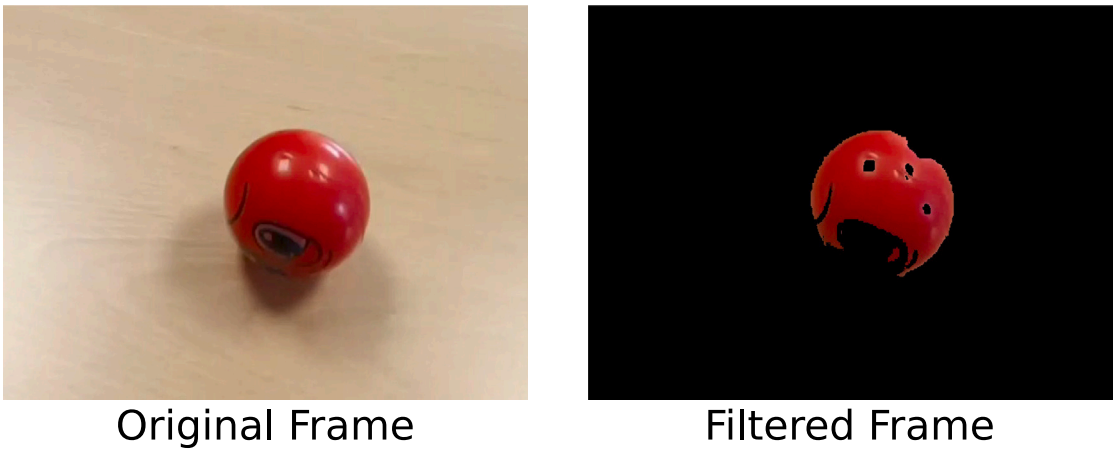


Fig. 1. Original frame and red-filtered frame resulting from the first step towards object detection. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.2.2. Color-based object detection module

This module shall detect a colored object for the robot to follow it. For this purpose, it must detect the object and estimate its distance and whether it is in the robot's direction. The original image comes from the frame buffer of the camera image capture module.

The color-based object detection module performs three tasks: color filtering, proximity calculation, and horizontal localization.

1. Color filtering: All the original image's pixels are filtered based on a color that can be configured by an input port. The filtering color is defined by a combination of the three RGB channels. Hence the detected object could be red, green, blue, yellow, cyan, magenta, or white.
   When a pixel does not pass through the color filter, it is saved as a black pixel in a second frame buffer, called *processed* frame buffer. On the contrary, if the pixel passes the filter, it is saved with its original color. We will call these pixels *colored pixels* (or *red pixels* when the filter is red, which will be the case in most of the examples). Therefore, the image saved in the processed frame buffer will be black except for the colored pixels. Fig. 1 shows the original captured frame on the left and the filtered image to detect the red ball on the right.
   It is worthwhile mentioning that the object to be detected should be of a different color than the background, and there should not exist a second object with the same color. When these assumptions do not hold, the results will not be reliable.
2. Proximity calculation: To calculate the object's proximity, every colored pixel of each frame is counted, and a proximity estimation is calculated based on the object size in the image. The fewer colored pixels found, the more distant the object will be. On the other hand, if there were many colored pixels, it would mean that the detected object is very close. Fig. 2 shows two examples: one of a distant object (low proximity on the left) and a closer object (high proximity on the right).
   The count of all colored pixels is scaled down to a 3-bit unsigned signal named *proximity*, whose value is proportional to the nearness of the object. That is, a value of seven ("111") means that the object is very close; on the other hand, when the object is far, it will be one ("001"). If no object is detected, its value will be zero. The proximity values of Fig. 2 are an example and could be changed depending on the object size. Moreover, the resolution of this signal can be easily increased (although for all the experiments carried out, three bits provide enough accuracy).
3. Horizontal localization: the object's horizontal localization must be determined to steer the robot towards the object. For this purpose, the frame is divided into eight vertical bins, and then a histogram of the colored pixels is obtained. Fig. 3 shows an example of the histogram obtained from a filtered frame. The histogram's most populated bin will determine the object's estimated horizontal localization. Although this is a simplification, if the object is rounded, the histogram will be unimodal (i.e., it will have just one peak); consequently, the most frequent bin will correspond to the horizontal localization of the detected object. We define the *centroid* as an 8-bit signal, having the value "one" for the bin number with the highest value and the value "zero" for the rest of the bins. In the example of Fig. 3, bin number five is the most frequent, indicating that the object is on the camera's right side. In this case, the centroid will have its bit number five to one and the rest of the bits to zero. Generally, the centroid will only have one bit to one and the other bits to zero, indicating where the object's centroid is in the horizontal plane. Nevertheless, there are two exceptions:
   - When the object is completely centered (i.e., the third and fourth bins have approximately the same frequency), the centroid will have the central two bits (3rd and 4th bits) to one.
   - When no object is detected, that is, when the number of colored pixels is below a minimum threshold, all the bits of the centroid will be zero.

Due to the aforementioned simplifications (i.e., just one colored object distinct from the background and rounded), this object detection module is simple and provides results with almost no latency to the frame reception. Furthermore, no additional memory is needed. Nevertheless, the module would need to be refined in more challenging environments.

3.2.3. Motor controller module

The GoPiGo robot used in these experiments is a differential wheeled robot, that is, it has two wheels that can be independently powered and controlled. For the robot to follow the colored object, the motor controller needs to know the object's proximity and its horizontal localization with respect to the robot (centroid), which is the information provided by the previous module.

On the one hand, the lower the proximity is, the faster the robot should go. On the other hand, when the object is near, the robot should decrease its speed. Moreover, when the object is too close, the robot should go backward. In addition, in the case that proximity is zero, it means that no object is detected, and therefore the robot should not move. In Fig. 2, the robot will go fast when proximity is 2 (left), but

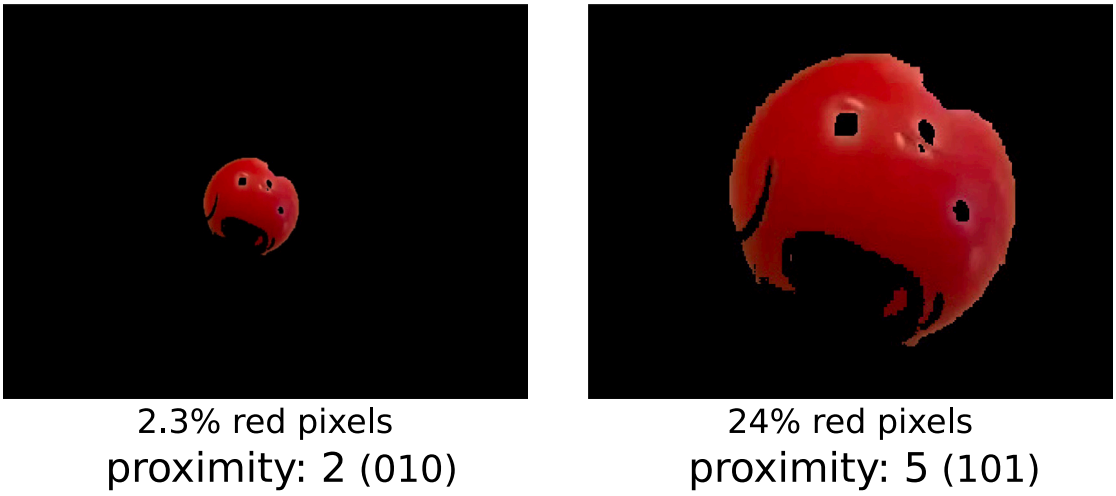


Fig. 2. Detection examples of a distant object (left) and a closer object (right). The ratio of red pixels determines the proximity. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

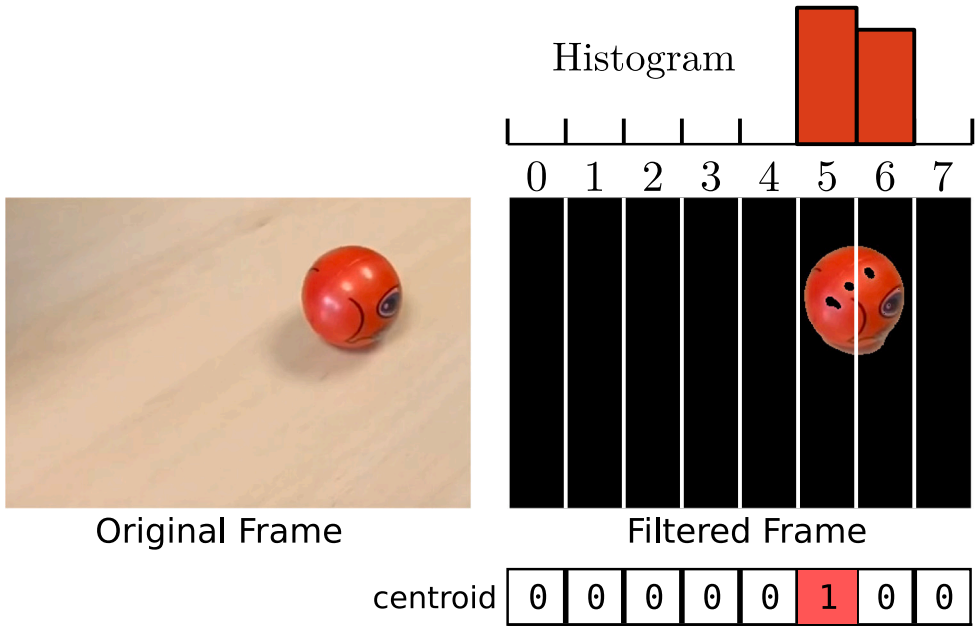


Fig. 3. Representation of the centroid calculation from the filtered image.

when proximity is 5 (right) it will stop, and for values of proximity larger than 5, the robot will go backward.

If the object is centered, as in Fig. 4(A), both wheels will be turned at equal speed. However, when the object is to the right (Fig. 4(B)), in order to steer the robot towards the object, the left wheel should be turned faster than the right one. Conversely, the right wheel should move faster when the object is to the left, as shown in Fig. 4(C).

As mentioned in the previous subsection, the centroid is an 8-bit signal, and the goal is to keep the object at the center, i.e., to maintain the central two bits of the centroid high (3rd and 4th bits). In Fig. 4, the robot will steer to the left when the centroid is "0100 0000" (object to the left), when the centroid is "0001 1000" (centered object) it will stop steering, and for a value of "0000 0010" (object to the right) of the centroid, the robot will steer to the right.

For the sake of simplicity, we have implemented a proportional control, which only considers the detection of the object at the present time; further improvements and any other closed-loop control system could be applied. The speed control is defined in Eqs. (1) and (2), where v_{left} and v_{right} represent the speeds of the left and right wheel, respectively; k_{magn} is the proportional constant for the speed magnitude (proximity); k_{steer} is the proportional constant for steering the robot depending on the object horizontal localization (centroid). The $convertedCentroid$ is a number considered zero when the object is centered (Fig. 4(A)), positive when the object is on the right (Fig. 4(B)), and negative when on the left (Fig. 4(C)).

$$v_{left} = \begin{cases} k_{magn} \cdot (5 - proximity) + k_{steer} \cdot convertedCentroid & if\ \ proximity > 0 \\ 0 & if\ \ proximity = 0 \end{cases} \tag{1}$$

$$v_{right} = \begin{cases} k_{magn} \cdot (5 - proximity) - k_{steer} \cdot convertedCentroid & if\ \ proximity > 0 \\ 0 & if\ \ proximity = 0 \end{cases} \tag{2}$$

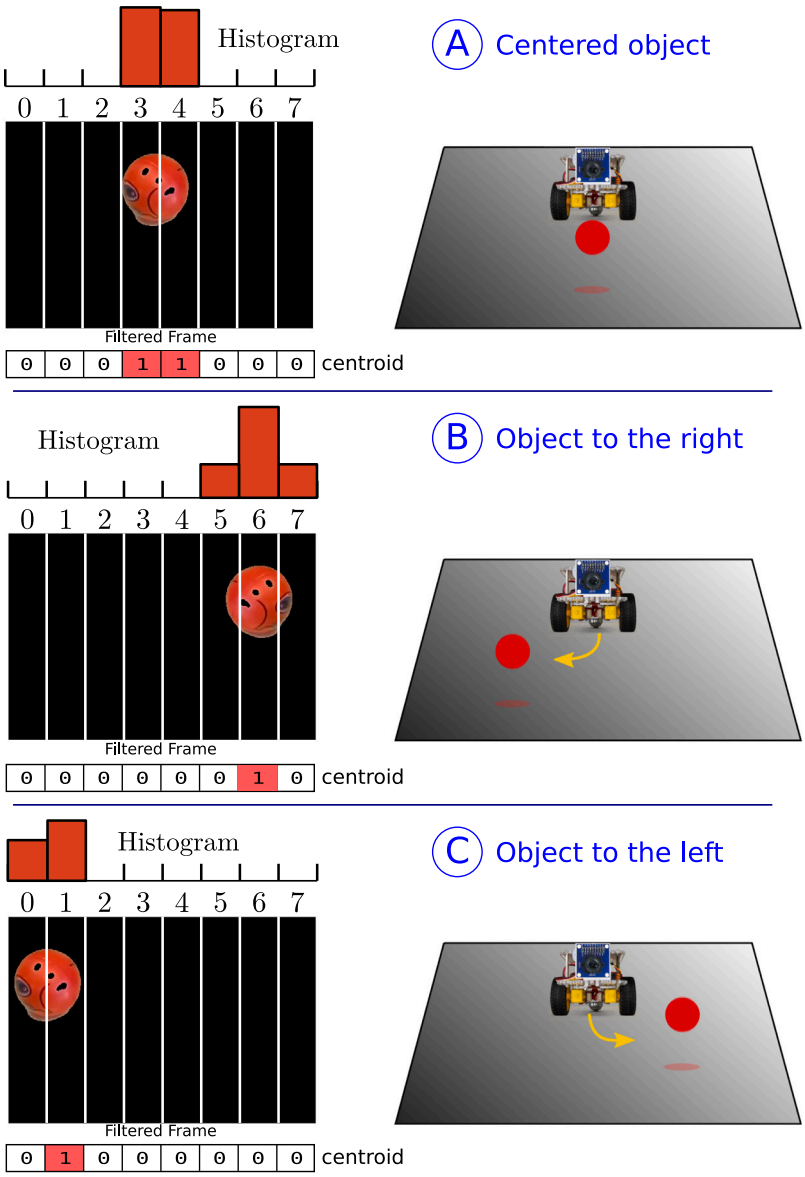


Fig. 4. Representation of the filtered frame, its histogram and centroid, and the motor control for the robot for different relative positions of the object. (A) when the object is centered. (B) when the object is to the right. (C) when the object is to the left.

The adjustment of the constants k_{magn} and k_{steer} in (1) and (2) should consider the robot's characteristics and the object dimensions. In addition, a more complex control, such as proportional–integral–derivative (PID) control, would not be difficult to implement.

3.2.4. GoPiGo SPI interface module

The motor controller detailed in the previous subsection generates the desired speed values of the motors. If the robot moves on an even horizontal surface with no slopes and no obstacles, the speed value can be directly matched with the motor voltage through PWM (Pulse Width Modulation). The PWM converter is a simple module that is also available at the library. However, in order to use the power electronics board of the robot, we have used the GoPiGo robot board, which allows driving each motor through Serial Peripheral Interface (SPI). This board can receive either the motor PWM duty cycle or the motor speed in degrees per second. It should be noted that this module is also the driver for the left and right motor of the robot, as well as for the servo motor (control of the servo is explained in the next subsection).

This module implements the SPI communication protocol and a Finite State Machine for sending SPI commands to the GoPiGo controller. Therefore, this interface also allows controlling the rest of the robot peripherals, such as sensors, LEDs, and servos.

3.2.5. Pan servo controller module

This Verilog module controls a 180° servo motor for tracking an object from a camera on a turret. The module receives the centroid from the color-based object detection module (Section 3.2.2) and provides the angular position of the servo. This is the format required for the GoPiGo robot to be sent through SPI (Section 3.2.4), but it can be easily mapped to any other format.

The controller makes the servo rotate at an angle proportional to the perceived centroid position; however, since the servo has a 180° range, once it has reached one of the limits, it will have to stay at that end even if the object continues moving away. As a result, when the camera is mounted on a servo-driven turret, it will automatically pan following the object until it goes beyond the servo limits (−90° to 90°).

3.2.6. VGA display module

Finally, we have included a VGA interface in the library, which is useful for debugging during development. The VGA interface can

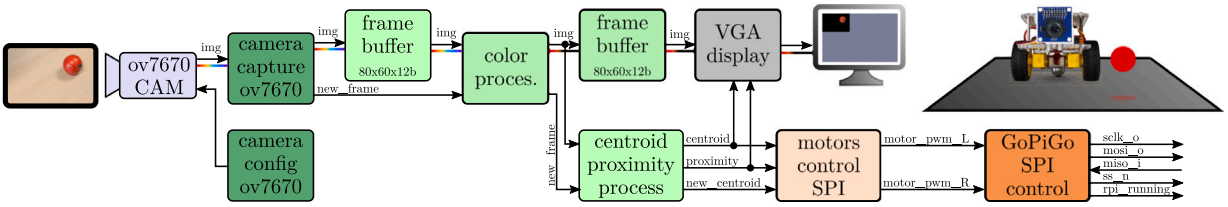


Fig. 5. Architecture of the monocular vision-based object follower robot.

simultaneously display up to three cameras. Since the object detection module (Section 3.2.2) can be switched on and off, the original and processed video can be visualized to check if the color processing module is working as expected. Moreover, the display shows the centroid and proximity bar graphs.

This module is composed of two sub-modules: the block in charge of the VGA synchronization signals and the block in charge of reading from the memory to display on the screen. The timing of the control and data signals have been defined for a resolution of 640×480 at approximately 60 Hz.

4. Experimental validation

In order to experimentally validate the proposed FPGA framework, including both the open-source toolchain and the developed open Verilog library, three robotics demo applications for a real vision-based robot have been created and successfully tested.

The GoPiGo mobile robot has been used to perform the experimental validation. It includes an onboard Raspberry Pi and the GoPiGo 3 expansion board, which integrates a microcontroller (Atmega328) to drive the robot sensors, servos and motors. This GoPiGo 3 board provides an SPI interface that allows controlling the motors or checking the status of the sensors, among other functions.

Instead of using the RaspberryPi CPU and programming the robot with software, we have substituted them with open FPGA development boards, Verilog HDL, and open development tools. We have used two different FPGA boards: the Alhambra-II [51] and the ULX3S-03 [52], which integrate Lattice FPGAs models of the iCE40 and the ECP5 family, respectively. The main difference between them lies in the number of logic resources available, including the internal memory blocks (BRAM) and the number of inputs/outputs.

In all the demo applications the robot's brain is implemented on an FPGA, instead of a microcontroller. This approach is not the classic one, since the latter is based on software and the former on hardware. Furthermore, the microcontroller processing is sequential, whereas each block of an FPGA is executed simultaneously, allowing parallel processing in each clock cycle.

4.1. Monocular vision-based object follower robot

The first experiment uses the Alhambra-II FPGA board in place of the RaspberryPi of the GoPiGo robot. A fixed ov7670 camera for tracking the colored object has been incorporated into the robot. Fig. 5 shows the block diagram that has been implemented on the FPGA. It combines several modules of the proposed `FPGA-Robotics` Verilog library.

As it can be observed, the information flows from left to right. First, a frame is captured with the ov7670 camera. Second, the captured image is filtered according to the selected color. Besides, the proximity and centroid are computed. Third, based on the relative position of the object and its proximity to the robot, the desired speeds of the motors are calculated. Finally, the proper SPI commands are sent to the robot. It should be noted that although the VGA display block has been included, it is only used for debugging. It is worthwhile mentioning that the processing is done in parallel as each module processes its information simultaneously. Therefore, there is no computational overhead when adding new modules, and the processing is achieved a few clock cycles after each frame is received. The limitation comes from the usage of finite FPGA resources, mainly the internal memory blocks of the FPGA. This is the reason why the frame size of this application has been reduced to 80×60, since the Alhambra-II FPGA has limited internal memory resources. The second limitation of this FPGA board is the reduced availability of I/O ports.

A demonstration of the correct functioning of this experiment can be seen in this video.[1]

4.2. Multi-camera object follower robot

Video processing is a computation-intensive activity; hence, performing at high frame rates or spatial resolutions normally overwhelms the processor capacity. The simultaneous video processing of two cameras would be difficult to implement in the original RaspberryPi of the robot even if it had a second camera port. However, the concurrent nature of FPGA processing eliminates the computational overhead. As mentioned in the previous subsection, this is possible as long as the FPGA has enough resources.

We have added a second camera to the previous design to demonstrate that an FPGA allows increasing the video processing capabilities without diminishing the performance. The FPGA should have enough resources for the task; therefore, we have replaced the Alhambra-II board with the ULX3S-03 board. The Alhambra-II board does not have enough I/O pins to connect a second ov7670 camera. In addition, the ULX3S-03 FPGA has around 30 times more BRAM than the Alhambra-II.

Fig. 6 shows the architecture of the FPGA design to follow an object using two cameras. The usage of two cameras increases the field of view (FOV) of the robot. As can be observed, each camera's capture and image processing is performed independently in parallel. Once both images have been processed with the color filter, unified values of the centroid and proximity of both cameras are calculated. As these unified values have the same format as with a single camera, the rest of the modules can be reused since the received values are identical.

The centroid calculation module is slightly different from the one explained in Section 3.2 since it requires combining the information from both cameras. Fig. 7 shows an example of determining the centroid with two cameras. The unified centroid is calculated from the maximum value obtained in both histograms. Since both images are oriented to increase the FOV, the cameras should not overlap. If the object is located at the edges (left or right), the resulting centroid activates the ending bit at the side where the object is detected.

4.3. Multicamera follower and tracker robot

To further prove the capabilities of parallel processing, we have added a third camera to track an additional second object concurrently. Therefore, two cameras are used to increase the FOV of the robotic platform, such as in the previous example, and the third one is mounted on a servo-driven turret at the rear of the robot to track another object

[1] https://youtu.be/rbdQ36ZJ7Lo

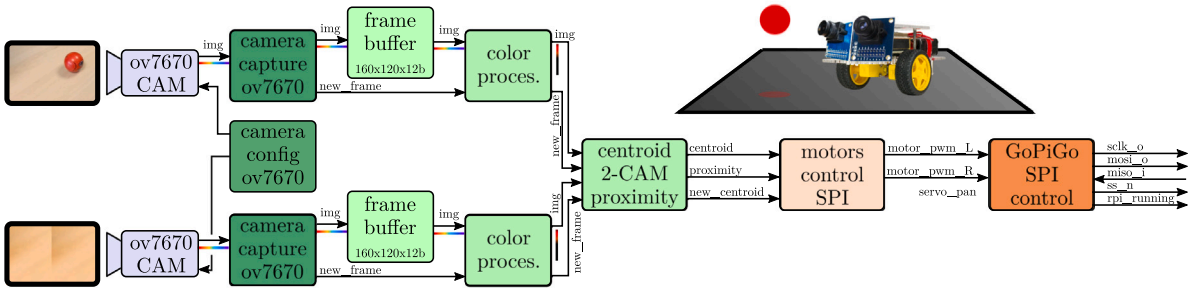


Fig. 6. Architecture of the two-camera object follower robot.

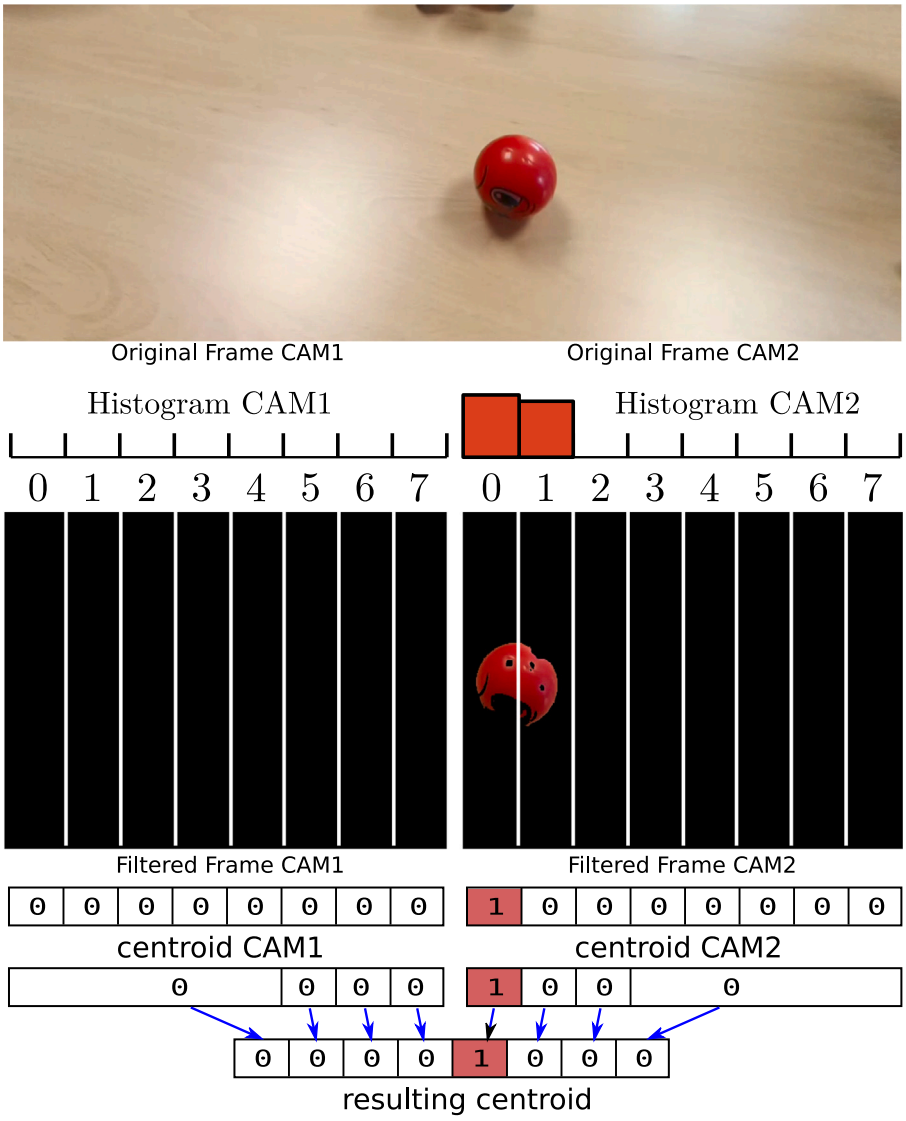


Fig. 7. Centroid processing with two cameras.

simultaneously. Fig. 8 shows the FPGA design architecture for the object follower and tracker application using three cameras.

In this case, three frame processing lines run simultaneously on the FPGA. The two cameras used to increase the FOV have the same layout used in the previous case. On the other hand, the third processing line tracks the second object with a servomotor-driven camera turret at the rear of the robot.

In this example, the only new block is the servo control. In this case, the control is adapted for servo-driven pan object tracking. A demonstration of the correct functioning of is experiment can be seen in this video.[2]

[2] https://youtu.be/6YHPovFdMn4

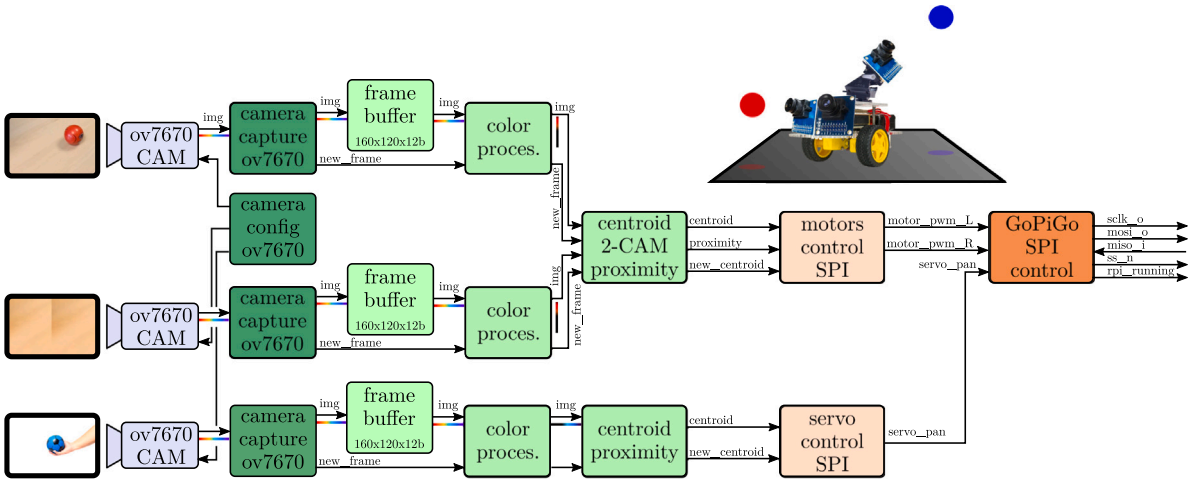


Fig. 8. Architecture of the three-camera object follower and tracking robot.

Table 1
FPGA resource consumption of the different experimental applications for the Lattice ECP5 85F FPGA (ULX3S FPGA board) using the open toolchain for FPGAs.

Resource	Single camera (res. 320 × 240)	Two cameras (res. 160 × 120)	Three cameras (res. 160 × 120)
RAM 16K	114 (54%)	60 (28%)	90 (43%)
Logic Slice	1846 (4%)	3395 (8%)	4611 (11%)
IO Slice	43 (21%)	58 (28%)	69 (34%)

Table 2
FPGA resource consumption of the different experimental applications for the Lattice ECP5 85F FPGA (ULX3S FPGA board) using the Lattice Diamond software.

Resource	Single camera (res. 320 × 240)	Two cameras (res. 160 × 120)	Three cameras (res. 160 × 120)
RAM 16K	192 (92%)	97 (46%)	133 (63%)
Logic Slice	437 (1%)	912 (2%)	2208 (5%)
IO Slice	43 (21%)	58 (28%)	69 (34%)

4.4. FPGA resource consumption

The FPGA resource consumption of the different experimental applications using the open toolchain for FPGAs is shown in Table 1. In order to adequately compare the percentages of the resources used, this comparison has been performed using the same physical FPGA. Since the Alhambra board does not have enough I/O pins for the second and third experiments, for this comparison, we have used the ULX3S-03 board, which includes a Lattice ECP5 85F FPGA.

Only non-zero resources are shown for all the resources available on the FPGA. It is worth noting that the monocular object follower has twice the resolution of the other scenarios. It can be observed how the resolution increment affects the FPGA memory resources (RAM 16K). With a single camera, a 320×240 resolution results in 54.81% of memory blocks. On the other hand, decreasing the resolution to 160×120 with two cameras uses only 28.85% of the memory blocks, and with three cameras 43.27%. Finally, it should be noted that since using three cameras only requires about 50%

4.5. Comparison with state-of-the-art FPGA toolchains

In order to validate and compare the used open toolchain for FPGA development with other FPGA proprietary toolchains, the bitstreams of the three presented robotics applications have also been generated with the toolchain provided by Lattice. In addition, the experiments have also been synthesized in FPGAs from other vendors, such as Xilinx and Intel-Altera.

4.5.1. Results using lattice diamond

Table 2 shows the resource consumption of the different experimental applications using the Lattice Diamond software. To adequately compare with the open toolchain for FPGA, the experiments have been implemented using the same FPGA (Lattice ECP5 85F). Only non-zero resources are reported.

Fig. 9 shows graphically the percentage of resources required for each of the experimental applications comparing the open-source toolchain with Lattice Diamond. The results obtained with Lattice software use the area optimization option for synthesis and place and route. The used resources were considerably larger for other synthesis strategies, such as balanced or speed optimization.

We can observe that there are considerable differences in how internal memory (RAM 16K) and logic resources are used by both tools. On the one hand, the total FPGA memory resources allocated by the open source tool are less than 55% for all three experiments (first row of Table 1). Conversely, the number of memory blocks allocated by Lattice Diamond software is around 50% higher than in the open-source tool. This is especially critical for the single-camera experiment, in which 92% of the FPGA memory blocks are used (see Table 2).

On the other hand, Diamond Lattice software uses considerably fewer logic resources, between 25% and 50% of the resources used in the open toolchain. Since the experiments are vision-based, the reduced usage of memory blocks is a clear advantage of the open tool, because larger frames can be processed, or even additional cameras could be included. On the other hand, the implemented algorithms do not entail high-complexity processing, and consequently, not many logic resources are needed.

4.5.2. Results for other FPGA vendors

The three robotics applications have also been implemented with both Vivado from Xilinx and Quartus from Intel-Altera. Since their underlying technology is different, the results cannot be directly compared.

For the Xilinx FPGAs, the development platforms used are Nexys 4 DDR, which has an Artix-7 FPGA; and the Zybo Z7-20, which has the Zynq-7000 SoC. The consumed resources are shown in Table 3.

The structure of the Xilinx FPGA resources is different from those of Lattice FPGAs. In this case, the capacity of the Xilinx FPGA memory blocks is 36K, instead of the 16K of the Lattice FPGAs. In addition, the Xilinx memory blocks can be used as one 36K memory block or two 18K memory blocks. In addition, each Xilinx slice includes 4 Look-up Tables (LUTs) and 8 Flip-Flops (FFs), whereas Lattice slices have 2 LUTs and 2 FFs. As a consequence, the resource consumption shown in Table 3 is

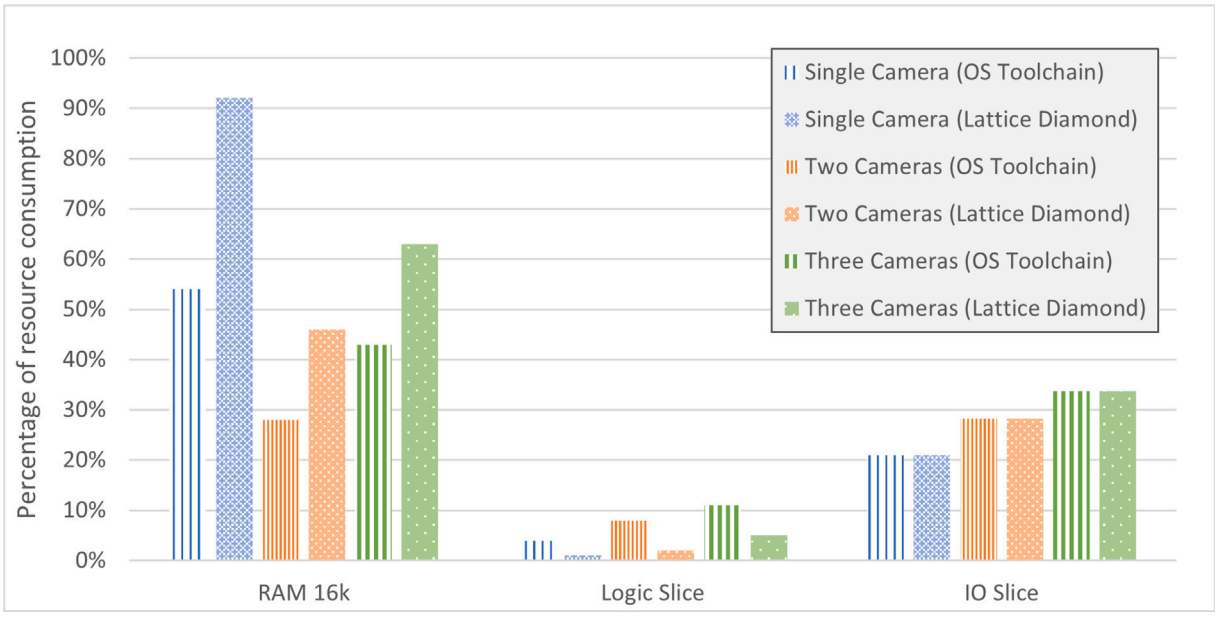


Fig. 9. Comparative graph showing the percentage of resources used in the three experimental applications with the open-source toolchain and the Lattice Diamond software.

Table 3
FPGA resource consumption of the different experimental applications for the Xilinx Nexys4 DDR and Zybo Z7-20 development platform using the Xilinx Vivado software.

	Resource	Single camera (res. 320 × 240)	Two cameras (res. 160 × 120)	Three cameras (res. 160 × 120)
Nexys4 DDR	Slice LUTs	463 (0.87%)	463 (0.87%)	2066 (3.88%)
	Slice Registers	370 (0.35%)	799 (0.75%)	1949 (1.83%)
	BRAM 36K	72 (51.43%)	48 (34.29%)	66 (47.14%)
	Bonded IOB	43 (34.40%)	58 (46.40%)	69 (55.2%)
	Resource	Single camera (res. 320 × 240)	Two cameras (res. 160 × 120)	Three cameras (res. 160 × 120)
Zybo Z7-20	Slice LUTs	519 (0.98%)	973 (1.83%)	2066 (3.88%)
	Slice Registers	426 (0.40%)	802 (0.75%)	1958 (1.84%)
	BRAM 36K	72 (51.43%)	48 (34.29%)	66 (47.14%)
	Bonded IOB	43 (34.40%)	58 (46.40%)	69 (55.2%)

Table 4
FPGA resource consumption of the different experimental applications for the Intel Cyclone 10 LP (10CL080YF484I7G) using the Intel-Altera Quartus software.

Resource	Single camera (res. 320 × 240)	Two Cameras (res. 160 × 120)	Three Cameras (res. 160 × 120)
Memory bits	1 843 200 (66%)	921 600 (33%)	1 267 200 (45%)
Logic elements	861 (1%)	1598 (2%)	3840 (5%)
IO pins	43 (15%)	58 (20%)	69 (24%)

lower than those shown in Tables 1 and 2 because the resource capacity of the individual blocks is higher. Therefore, a direct comparison would not be sensible since the technologies are different.

Finally, the three robotics applications have also been synthesized with the Quartus software from Intel-Altera. The development platform used is the Intel Cyclone 10 LP (10CL-080YF484I7G) FPGA. The consumed resources are shown in Table 4. It should be noted that the resource consumption obtained for the memory is reported in bits and not as the number of available elements as in the previous cases.

The number of logical elements represents a percentage of 5% in the example of the three cameras. On the other hand, the FPGA memory resources consumed is equal to 45%, corresponding to the number of bits used in each frame buffer of the design, with 12 bits per pixel.

5. Conclusions

The actual situation of open reconfigurable computing is similar to that of the open-source GNU Compiler Collection (GNU toolchain) for software development decades ago. Its standardization, support of different processor architectures, and adoption brought many open source libraries (both general and field-specific), more reutilization, and fostered software development in many application fields. The standardization, the support of many FPGA boards, and the adoption of an open reconfigurable computing toolchain (perhaps F4PGA) will possibly bring similar benefits to the FPGA community.

The current main drawback of the open approach is that most advanced hardware FPGA platforms are not supported yet; they are only available for use in conjunction with the proprietary toolchains of the manufacturers. Nevertheless, the number of supported boards in the open FPGA ecosystem is continuously growing (currently, it targets the Xilinx 7-Series, Lattice iCE40, Lattice ECP5 FPGAs, and QuickLogic EOS S3). Besides, it is gradually being expanded to support more high-performance platforms from several manufacturers.

In this context, the main contribution of this work is an open-source Verilog library for vision-based robot applications, named `FPGA-Robotics`, aiming to help developers to benefit from the fast execution speed, power efficiency, and reconfigurability of FPGAs when

implementing vision-based robotic applications. It includes a sensor driver (camera), an actuator driver (GoPiGo motors and servo), perception blocks (color filtering and image processing), control blocks (motor controller, servo controller), and a debugging block (VGA display).

The usage of `FPGA-Robotics` shortens the development time as some of their Verilog blocks may be integrated into the applications without developing them from scratch. It also helps to introduce trustworthy source code in the applications, as the library code is publicly available, may be inspected, and has been typically tested widely by the community. In addition, `FPGA-Robotics` is open-source, and its modules can also be freely used in combination with popular proprietary FPGA toolchains of widespread FPGA vendors (Lattice/Xilinx(AMD)/Intel-Altera).

All library blocks have been experimentally validated by the implementation of three vision-based reactive applications for a real robot, making use of an open toolchain that is based on Verilog language, YoSys synthesis, nextpnr place-and-route tool, and OpenFPGALoader (all of them managed with the open-source APIO tool). In these applications, the GoPiGo robot is able to follow a moving colored ball using one camera (or two cameras for a wider FOV). This behavior has been easily extended to concurrently track a different colored object with a third camera on top of a servo. The parallel nature of FPGAs facilitates it. In all cases, the robot brain, with all the image processing and the control decisions, is fully deployed at the onboard FPGA board. The three applications are publicly available so that they may be replicated and inspected.

A significant contribution of this work is the comparison of the resource consumption of all the developed applications for several open/Lattice/Xilinx(AMD)/Intel-Altera toolchains and boards, as shown in Section 4.5. The experimental results show that, depending on the nature of the application, the open toolchain has no disadvantage in resource usage. Concretely, in the experiments presented, it performs better on the same Lattice hardware than the proprietary toolchain regarding the FPGA BRAM memory blocks while performing worse concerning the logic blocks.

Since verifying FPGA vision-based systems for robotics is a slow and challenging endeavor, as future lines, we are working to integrate fast open simulators such as Verilator with the Gazebo robotic simulation environment. This simulated environment will help set the different parameters present in several library modules. In addition, we intend to develop more complex robotics applications, such as those based on Finite State Machines, including the enable-disable functionality in the Verilog blocks of the `FPGA-robotics` library. Finally, we are developing more blocks to support external memories, other sensors (such as IR, US, or RPLIDAR), and other robots (such as the TurtleBot2).

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.

Data availability

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

Acknowledgments

This research was partially funded by the Community of Madrid in the framework of the research project, Spain RoboCity2030-DIH-CM (2019–2022): RoboCity2030-Madrid Robotics Digital Innovation Hub, Spain, Programa de Actividades de I+D entre Grupos de investigación de la Comunidad de Madrid en Tecnologías 2018 project, Spain ref. S2018/NMT-4331.

References

[1] J. Ruiz-Rosero, G. Ramirez-Gonzalez, R. Khanna, Field programmable gate array applications—A scientometric review, Computation 7 (4) (2019) 63.
[2] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, M. Milanovic, Yosys+ nextpnr: an open source framework from verilog to bitstream for commercial FPGAs, in: 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM, IEEE, 2019, pp. 1–4.
[3] A. Romanov, M. Romanov, A. Kharchenko, FPGA-based control system reconfiguration using open source software, in: 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering, EIConRus, IEEE, 2017, pp. 976–981.
[4] H. Yu, H. Lee, S. Lee, Y. Kim, H.-M. Lee, Recent advances in FPGA reverse engineering, Electronics 7 (10) (2018) 246.
[5] D. Celebucki, S. Graham, S. Gunawardena, Reversing a lattice ECP3 FPGA for bitstream protection, in: International Conference on Critical Infrastructure Protection, Springer, 2018, pp. 91–111.
[6] T. Zhang, J. Wang, S. Guo, Z. Chen, A comprehensive FPGA reverse engineering tool-chain: From bitstream to RTL code, IEEE Access 7 (2019) 38379–38389.
[7] J. Arroyo, J. González, APIO open source ecosystem for open FPGA boards, 2016, https://github.com/FPGAwars/apio.
[8] J. Arroyo, C. Venegas, J. González, IceStudio visual editor for open FPGA boards, 2017, https://github.com/FPGAwars/icestudio.
[9] YosysHQ, OSS CAD suite, 2022, URL https://github.com/YosysHQ/oss-cad-suite-build.
[10] C. Alliance, FOSS flow for FPGA (F4PGA) work group, 2022, https://f4pga.org/.
[11] M. Quigley, B. Gerkey, W.D. Smart, Programming Robots with ROS: A Practical Introduction To the Robot Operating System, O'Reilly Media, Inc., 2015.
[12] Open Robotics Foundation, ROS 2 hardware acceleration working group, 2021, https://github.com/ros-acceleration.
[13] F. Machado, R. Nieto, F.-C. J., L. D., C.J. M., FPGA robotics, 2022, https://github.com/JdeRobot/FPGA-robotics.
[14] D. Shah, Nextpnr - a portable FPGA place and route tool, 2022, URL https://github.com/YosysHQ/nextpnr.
[15] J. Ordóñez Cerezo, E. Castillo Morales, J.M. Canas Plaza, Control system in open-source FPGA for a self-balancing robot, Electronics 8 (2) (2019) 198.
[16] J.M. Cañas, J. Fernández-Conde, J. Vega, J. Ordóñez, Reconfigurable computing for reactive robotics using open-source FPGAs, Electronics 11 (1) (2021) 8.
[17] 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.
[18] 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.
[19] 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.
[20] 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.
[21] K. Guo, S. Zeng, J. Yu, Y. Wang, H. Yang, A survey of FPGA-based neural network accelerator, 2017, arXiv preprint arXiv:1712.08934.
[22] T. Wang, C. Wang, X. Zhou, H. Chen, An overview of FPGA based deep learning accelerators: challenges and opportunities, in: 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems, HPCC/SmartCity/DSS, IEEE, 2019, pp. 1674–1681.
[23] D. Gschwend, Zynqnet: An FPGA-accelerated embedded convolutional neural network, 2020, arXiv preprint arXiv:2005.06892.
[24] 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.
[25] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, T. Delbruck, DeltaRNN: A power-efficient recurrent neural network accelerator, in: Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2018, pp. 21–30.
[26] J. Zhang, J. Li, Improving the performance of opencl-based FPGA accelerator for convolutional neural network, in: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017, pp. 25–34.
[27] J. Li, K.-F. Un, W.-H. Yu, P.-I. Mak, R.P. Martins, An FPGA-based energy-efficient reconfigurable convolutional neural network accelerator for object recognition applications, IEEE Trans. Circuits Syst. II 68 (9) (2021) 3143–3147.
[28] 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.
[29] 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.
[30] S. Murray, W. Floyd-Jones, Y. Qi, D.J. Sorin, G. Konidaris, Robot motion planning on a chip., in: Robotics: Science and Systems, 2016.
[31] A. Tuncer, M. Yildirim, Design and implementation of a genetic algorithm IP core on an FPGA for path planning of mobile robots, Turk. J. Electr. Eng. Comput. Sci. 24 (6) (2016) 5055–5067.

[32] 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.
[33] 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.
[34] 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.
[35] B.L. Sharma, N. Khatri, A. Sharma, An analytical review on FPGA based autonomous flight control system for small UAVs, in: 2016 International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT, IEEE, 2016, pp. 1369–1372.
[36] 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.
[37] J. Caro, A. Barrientos, E. Mayas, Hybrid bio-inspired architectura 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.
[38] 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.
[39] 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.
[40] 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.
[41] J. Lee, H. Chen, J. Young, H. Kim, RISC-V FPGA platform toward ROS-based robotics application, in: 2020 30th International Conference on Field-Programmable Logic and Applications, FPL, IEEE, 2020, p. 370.
[42] D. Koch, D. Ziener, F. Hannig, FPGA versus software programming: Why, when, and how? in: FPGAs for Software Programmers, Springer, 2016, pp. 1–21.
[43] E. Klingman, FPGA programming step by step, Embedded Syst. Program. 17 (4) (2004) 29–37.
[44] N. Tredennick, B. Shimamoto, The inevitability of reconfigurable systems, Queue 1 (7) (2003) 34–43.
[45] N. Tredennick, The case for reconfigurable computing, Microprocess. Rep. 10 (10) (1996) 25–27.
[46] C. Wolf, Yosys open synthesis suite, 2016.
[47] C. Wolf, M. Lasser, Project icestorm, 2015, http://www.clifford.at/icestorm.
[48] D. Shah, Project trellis, 2022, URL https://github.com/YosysHQ/prjtrellis.
[49] G. Goavec-Merou, Open FPGA loader, 2022, URL https://github.com/trabucayre/openFPGALoader.
[50] Modular Robotics Incorporated, GoPiGo robot, 2020, URL https://gopigo.io/.
[51] J. Arroyo, C. Venegas, J. González, Alhambra-II FPGA, 2018, https://github.com/FPGAwars/Alhambra-II-FPGA.
[52] Radiona.org, ULX3S FPGA, 2022, https://radiona.org/ulx3s/#home.

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.

Rubén Nieto has received his Ph.D. degree (with honors) 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 are in Mobile Robots, Multisensor Integration, Multiprocessor System-on-Chip and Digital and Embedded Systems.

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.