Hardware-in-the-loop simulation with dynamic partial FPGA reconfiguration applied to computer vision in ROS-based UAV

Erwan Moréac, El Mehdi Abdali, François Berry, Dominique Heller, Jean-Philippe Diguet

Hardware-in-the-loop simulation with dynamic partial FPGA reconfiguration applied to computer vision in ROS-based UAV

Erwan Moréac*, El Mehdi Abdali**, Francois Berry**, Dominique Heller*, Jean-Philippe Diguet*
*LAB-STICC, CNRS, Université Bretagne Sud, 56100 Lorient, France
**Institut Pascal, Université Clermont Auvergne, 63178 Aubière, France

***Abstract*— Hardware in the loop simulation has become a fundamental tool for the safe and rapid development of embedded systems. Dynamically and partially reconfigurable FPGA provide an energy efficient solution for high performance computing in embedded systems, such as computer vision, with limited resources. Finally 3D simulation with realistic physics simulation is required by designers of Unmanned Aerial Vehicle (UAV) and related missions. The combination of the three techniques are required to design UAV with reconfigurable HW/SW embedded systems that can self-adapt to different mission phases according to environment changes. But they require different complex and specific skills from separated communities and so are not considered simultaneously. In this paper we demonstrate a complete framework that we apply to a UAV case simulated with the well adopted Gazebo 3D simulation tool including the Ardupilot model. According to usual practices in Robotics, we use Robot Operating System (ROS) middleware over Linux that we implement on a separated Intel Cyclone V FPGA board including HW/SW interfaces. As a convincing case study we implement, besides software classical navigation tasks, a vision-based emergency-landing security task and a detect and tracking classic mission application (TLD) that can run in different HW and SW versions dynamically configured on the FPGA according to mission steps simulated with Gazebo.**

I. Introduction

Security and reliability of autonomous vehicles, such as UAVs, require fast and accurate perception capabilities based in particular on image and radar sensors. Reactivity constraints increase with vehicle speed, moreover systems must be robust to network outage. This means that high performance embedded computing (HPEC) is required to guarantee short response times and safety.

Energy efficiency is a key parameter in embedded systems and FPGA are known to be efficient for the implementation of massively parallel pixel-level image processing tasks. Cost is another important parameter so large FPGA are usually not an appropriate solution. But depending on the mission phases, the tasks of the mission may change, moreover the time constraint also depend on the speed and location of the vehicle. So, designers can take benefit of the partial and dynamic reconfigurability (DPR) of FPGA to share hardware resources.

But the domain of Robotics is mainly based on a software and strongly relies on software reuse and open source to deal with the design of complex systems. ROS (Robot Operating System) is a typical example of such a methodology. This is a middleware widely used by roboticians to manage communications and interactions between tasks and between tasks and sensors.

(Submitted version)

Hybrid architecture (ARM+FPGA) can help to bring closer the two worlds but FPGA require specific skills, which are far from robotician concerns. Obviously the use of DPR is even more inaccessible.

Finally, the validation of such FPGA-based embedded systems is another challenge since it requires to combine tools from different domains. First designers need to simulate mission scenario and interactions with environment by means of a virtual world including physics engines. Then they need autopilot models for the low level control of the vehicle. At the software level, ROS is a well adopted solution for the rapid development of the embedded software, it must be considered in the simulation framework and this can be done with a Hardware In the Loop (HIL) approach. At the hardware level, there is today no real solutions available for the implementation of tasks on FPGA and so no use of DPR.

Considering the huge interest of using FPGA and DPR in embedded systems for robotics, we have developed a HIL environment that interacts with a standard robot simulator. Our approach is firstly based on a Hardware/Software architecture model including data transfers and task activation. Secondly it is compliant with the ROS environment. Finally it includes the control of DPR according to an architecture model based in reconfigurable tiles.

The remaining of the paper is organized as follows. Section II details the state of the art. Then we introduce our platform model in Section III. In Section IV we detail our UAV case study and the last section concludes the paper.

II. Related works

A. *Dynamic Partial Reconfiguration (DPR) with FPGA*

DPR has been widely studied since the advent of DPR-enabled FPGA in the early 90's. Many research works have addressed the problem of application implementations over DPR architectures. They are usually based on a reconfigurable systems structure composed of one static region that holds the unchanged design parts and one or more reconfigurable regions that hold the loaded-on-demand design parts. The Erlangen slot machine (ESM) [13] is an representative and complete example of such DPR architectures with slots equivalent to reconfigurable regions. The intra-chip communications have been performed over four different mechanisms, each one is adapted to a specific kind of data exchange: direct connections, crossbar, shared memory and Reconfigurable Multiple Bus (RMB) which are "a special through in regions switches" implemented in tiles. This relatively complex communication structure has been justified to resolve the common data exchange dilemmas and has been taken as a reference for several DPR architectures such as [8] where a slot-based DPR architecture has been proposed with an automatic floor-planning and BitStreams generation.

The architecture used an overhead-adaptive communication mechanism [7] that uses dynamically established nodes with respect to the number of on-the-fly added modules. Reconfigurable Datapath for dataflow applications is another type of DPR-based architecture. Ardoise [11] was an example of such approaches. It was, in contrast, a multiple FPGA platform where each one contains a single data path. This type of implementation was motivated by the limited-size of FPGA (case of first generations) or because dynamic reconfiguration was available but could not be partial. However, they were based on hardware task models that have been used in numerous of works. The same hardware task model with two buffers has been used in the definition of a reliable RTOS to handle the scheduling, allocation and reconfiguration of tasks within a DPR architecture [6].

With larger FPGA another type DPR architecture, based on NoC, has emerged. In [21] all identical slots are connected in the same way regarding each other, I/Os and shared resources which enhances the location-independent of reconfigurable hardware tasks. Following nearly the same structure, in RAMPSoC architecture [5] the reconfigurable modules are not only hardware accelerators but also can be different processor types and/or co-processors. This type of architecture can be considered as an adaptive MPSoC platform.

We consider these solutions and many other as examples of DPR architectures that may be embedded in UAVs. However, none of them has been considered in a global project that requires systematic interfaces with 3D simulation and a robot platform. Another point is that, the great majority of cases is based on Xilinx FPGA, whereas we consider Altera/Intel devices to be compliant with our industrial partners.

B. *Hardware In the Loop simulations for UAVs*

The first step to perform HIL simulation is the virtual world simulator. A reliable world modeling is crucial to perform accurate simulations compared to the reality. Besides, a realistic environment is also important to evaluate the computer vision efficiency. There already exists in the literature several simulators aiming these objectives. Gazebo [12] is one the most popular simulation platforms for the research work. It allows to create 3D worlds using a configurable physics engine where UAVs and sensor models can interact with it. RotorS [3] is another simulator based on Gazebo. This modular framework proposes HIL with a PixHawk board (PX4) and to design its own Micro Air Vehicle (MAV). Then, jMAVSim [1] is a simple simulator designed in order to test and validate the PX4 hardware and autopilot. Hence, the simulator has simple sensor models and graphic engine without any objects in the environment. Finally, AirSim [17] is a simulator which relies on the Unreal Engine for the visual and the physical simulation. It also supports the MavLink protocol and can be used for HIL simulations.

Previous works have addressed HIL simulations such as in [9]. In [14], the authors present HIL simulations including a PixHawk board. This work does not use ROS to communicate with Gazebo but rather a custom middleware created with Qt in C which limits its exploitation by other research and developer teams. In [15] they propose a setup for multi-UAV HIL simulations. Each UAV is composed of an Odroid board that are connected to Gazebo thanks to the use of ROS. Nevertheless, the architecture used in this paper still not use a FPGA to enable high performance computer vision. However, some works that combines the use of ROS with a FPGA have been presented. In [20], they design a FPGA component associated with ROS in order to accelerate an image labeling. The way it was implemented make the image processing 1.7 times faster than the software version. The same research team proposed in [19] cReComp, an automated design tool to improve productivity of ROS-compliant FPGA component. Then, authors of [16] proposed an architecture exploration for SLAM processing based on works of the two aforementioned papers.

Fig. 1. Picture of the target UAV.

From our knowledge, there is no contribution that combines UAV HIL simulations with DPR using FPGA. In this paper, we propose a platform setup to perform UAV HIL simulations with DPR for high performances computer vision. This framework also allows of non-experts to develop DPR architectures which improves the productivity to design this kind of architecture.

III. Platform proposition

In this section we first describe the targeted UAV platform for HIL simulations. Thereafter, we perform the overview of the software platform with each program to run the whole simulation within a host PC. Then, we show the UAV board we want to include in HIL simulations. Finally, we explain in detail how to perform HIL simulations with a system using dynamic partial reconfiguration in FPGA.

A. *Target UAV platform*

The robotic setup used is a full custom UAV hexarotor. As can be seen in Figure 1, the UAV is composed of six propellers powered by brushless motors. Each motor is associated with a motor controller and they are all driven by a PixHawk board. This board runs the ArduPilot autopilot and embeds a 32-bit ARM Cortex M4 processor with an IMU. The IMU contains a 16-bit gyroscope, a 14-bit accelerometer and a 14-bit compass. The PixHawk is also connected to a GPS and to an optical flow which computes the altitude and the UAV speed using a downward facing distance sensor and a downward facing camera.

For the purpose to make this UAV autonomous, it is equipped with sensors to detect obstacles. To reach this objective, six ultrasound sensors with a range from 0.2 to 6m are used as well as six short-range infrared sensors with a range from 0.1 to 1.5m and six long-range infrared sensors with a range from 0.8 to 5.5m. Figure 1 shows that each type of the aforementioned sensors are distributed equally in six sections covering the whole UAV. These sensors are

connected to a board with an Altera Cyclone V System-on-a-Chip (SoC). Thus, this board is able to compute the obstacle avoidance based on distance sensors data and simultaneously execute other software and hardware tasks. Thanks to this computation power, the Cyclone V board sends orders to the PixHawk board in order to drive the UAV using a serial link. The Cyclone V FPGA allows hardware acceleration for tasks such as computer vision. In our mission scenario it is used to search emergency landing areas and to perform tracking applications. The power supply of the two boards and peripherals in the UAV is provided by a 6000 mAh Lipo battery. Finally, a Datalink 2.4Ghz module is plugged to allow priority remote control for security reasons.

B. *Overview*

In order to perform HIL simulations, we first implement and execute the whole simulation loop on a PC. Figure. 2 shows the four main actors of a ROS-based UAV simulation. Firstly, ROS is the middleware that helps at developing the UAV board program. Secondly, Gazebo is the software that models the UAV and the virtual environment. Thirdly, the Software In The Loop (SITL) emulates the PX4 autopilot embedded in the Pixhawk board and provided a flight simulator to move the UAV in Gazebo. Fourthly, MAVROS makes the link between the ROS-based program and the UAV autopilot. The next subsections give more detailed information about aforementioned simulation actors.

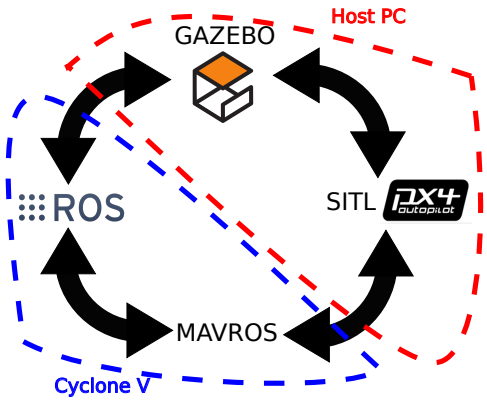

Fig. 2. Overview of the simulation loop. Everything can be executed on the host PC and to run in HIL, the ROS and MAVROS parts must be executed on the UAV board.

1) Robot Operating System (ROS): ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Every program in ROS is called a node. When using ROS, it creates a node called ROS master, it provides naming and registration services to the rest of the nodes in the ROS system. Every independent task can be separated into nodes which communicate with each other through channels. These channels are also known as topics. Every topic can have any number of broadcasters and receivers. This structure helps to separate different parts of the code into different modules, manage their builds and dependencies and develop them separately.

2) Gazebo: Gazebo [12] is a popular software in the robotics community and is completely open-source, so that users can easily define 3D virtual worlds, sensor models, and communication protocols. Thanks to the open dynamics engine (ODE), Gazebo can present a system model robot with high accuracy in real-time conditions. Besides, it can be applied forces and moments in the six-degrees-of-freedom UAV model. In order to gather sensors and video data from the virtual world, sensor and camera models are associated with plugins developed in C. The same principle is used to control UAV rotors with the autopilot.

3) Software In The Loop (SITL): The SITL simulator [18] allows to run Plane, Copter or Rover without any hardware. It is a build of ArduPilot autopilot on the host PC. It provides designers with a native executable that allows to test the behavior of the code without hardware. When running in SITL, the sensor data comes from a flight dynamics model in a flight simulator. The communication protocol used by ArduPilot is MAVLink for Micro Air Vehicle Link. This is a protocol for communicating with small unmanned vehicle. Hence, a bridge is required between the autopilot and ROS.

4) MAVROS: This is a ROS package that provides communication driver for various autopilots with MAVLink communication protocol. Thanks to this package, the ROS program developed is able to send orders and receive data from the autopilot.

As the entire software architecture has been described, we now present the UAV board architecture.

C. *UAV board architecture*

The board we embed in the UAV is composed of a Cyclone V SoC as mentioned earlier. The Hard Processor System (HPS) is composed of two processors ARM A9 and the Field-Programmable Gate Array (FPGA) fabric is an Arria V FPGA using a 28 nm technology. Both HPS and FPGA have access to a 1 GB DDR3 RAM. The next subsection brings more information about the reconfigurable architecture of the system.

D. *The partially reconfigurable architecture*

As depicted in Figure. 3, the FPGA fabric is divided in two reconfiguration regions. The use of two regions is due to the FPGA die size compared to the application needs in their HW versions. Each region can retrieve image data either from a 8 bits streaming image interface or from a 32 bits memory mapped interface connected to an external DDR shared memory. In addition to reconfigurable regions, the shared memory is accessed by the HPS for initializing the bitstreams and the reconfiguration manager. The reconfigurable regions have a width of 16 and a height of 77 giving a hardware resources amount of 10010 ALMs, 40040 Register, 38 DSPs and 154 M10K with a total memory capacity of 1576960 bits. The reconfiguration time is of around 5ms per region in $scrub$ mode and 6.6ms in and/or mode.

E. *Hardware In the Loop (HIL)*

In this subsection we describe first the software setup of the UAV to enable real experiments. Then, we show what are the necessary modifications to achieve HIL simulation with DPR.

1) UAV software setup: Figure. 4, inspired from [15], shows the UAV software setup. As mentioned earlier, we can see that the hexarotor is composed of two boards:

1) the PixHawk, a cortex-M4 based-board that embeds the autopilot,
2) a board with a Cyclone V SoC with ROS kinetic and Ubuntu 16.04 installed.

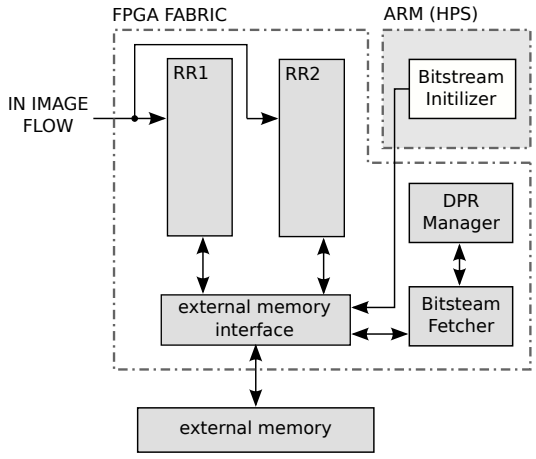

Fig. 3. Architecture model used in the experiments.

ROS allows us to exploit communication topics to connect most of the components within the Cyclone V. First of all, we have created a ROS node named mission manager. This node has to supervise applications selection in order to decide which application to prioritize. According to the mission manager requests in terms of application needs recovered from topics, another ROS node, called configuration controller, launches application in their HW or SW versions. If necessary the controller configures the FPGA by means of DPR to allocate resources according to application priorities. A synchronization mechanism is introduced to guarantee the end of a hardware task before any reconfiguration. Then, prioritized applications, like tracking for instance, can benefit from HW acceleration and reduce the CPU.

The ROS node sensor interface sends to the mission manager sensors raw data via topics. There are sensors data for the obstacle avoidance (from on-board sensors) mentioned in Section III-A. From these raw sensor data, a data fusion is performed to enhance obstacle detection reliability as described in [4]. The sensor interface also retrieves the two camera video flows.

ROS topics are used within the Cyclone V and the MAVLink communication protocol is employed through a serial link between the Cyclone V board and the PixHawk board. Thus, the mission manager is able to send orders such as waypoints to the PixHawk to drive the UAV. Similarly, the PixHawk sends sensors data and information about the UAV state such as the battery level and sensors state. A GPS and a px4flow are connected to the PixHawk by an USB serial link. The GPS gives the global positioning and the PX4flow provides the altitude and the UAV speed. We now present the HIL software setup and emphasize differences with the UAV setup.

2) HIL software setup: With the aim to carry out HIL simulations with the Cyclone V, we have to replace Cyclone V external connections. Indeed, on-board sensors and the two cameras are substituted by Gazebo and the PixHawk by the SITL software. The scheme in Figure. 5 illustrates the HIL software setup. A host PC is the new interlocutor of the Cyclone V, Ubuntu 16.04 and ROS kinetic are installed as well. Cyclone V and the host PC are connected via an ethernet link.

In order to allow communications with Gazebo, the sensor interface has been replaced by the node Gazebo interface. Hence, the board is able to retrieve data from Gazebo virtual sensors. In Gazebo, it is possible to model every sensors needed and they will react to the virtual world created in this software. Thus, obstacle detection sensors, cameras, IMU and so on are emulated and provide data through topics. These sensors can be configured to approach the characteristics of real ones. PixHawk related sensors data are sent to SITL and on-board sensors with cameras data are sent to the Cyclone V with topics. Sending data through a topic between two different machines is done by configuring the same ROS master node for both machines. In this way, the host PC and the UAV board may exchange data using ROS.

In order to recover PixHawk sensors data with SITL, Gazebo plugins allows SITL to exploit these sensors. It is important to remind that SITL emulates the autopilot ArduCopter embedded in the PixHawk. The connection between the Cyclone V remains a serial MAVLink link thanks to MAVROS and the SITL capabilities to reproduce the MAVLink communication protocol. SITL owns a feature that let us simulate sensors or UAV parts failures which is very interesting to evaluate adaptation abilities of the UAV program. Finally, the telemetry and the base station are not appearing in Figure. 5 since it is only used in real experiments to perform emergency landings, and as the UAV in HIL simulations moves in a virtual world, there is no harm in case of crashes during test phases.

Once the HIL setup is complete, we can perform experiments with several applications in order to validate the process using DPR.

IV. Case study

In this case study, we show an UAV performing computer vision while flying in a virtual world with the real SoC board. It allows to validate the FPGA reconfiguration system by switching the application executed in hardware. Thus, we first present applications that will be executed on the UAV board in software or in hardware and we explain how they work. Then, we show results from HIL simulations of the whole system.

A. *Detection and Tracking*

The first two co-processors considered in this study are the main blocks of the Tracking Detection Learning (TLD) tracking algorithm. The details of the co-processors, as well as their hardware implementations, are discussed in a previous paper dedicated to the architecture optimization to be disclosed in the final version.

1) Tracking: The TLD algorithm uses a median flow tracker [10] which is based on a grid of points where the Kanade Lucas Tomasi (KLT) method is applied individually to each of them. The points in a bounding box are firstly tracked with KLT in order get their displacements in the next frame, these displacements being called a *forward tracking*. After that, these points (with their neighborhood) are tracked inversely to the original frame to assess the capability of the algorithm to find back their original position. The points with a low forward-backward error (difference between original positions and predicted original positions) and a high similarity between original and tracked patches are considered as reliable and their tracking is used to retrieve the scale and object displacement. The remaining points are considered as outliers. The similarity between the original and predicted patches is assessed using the Normalized

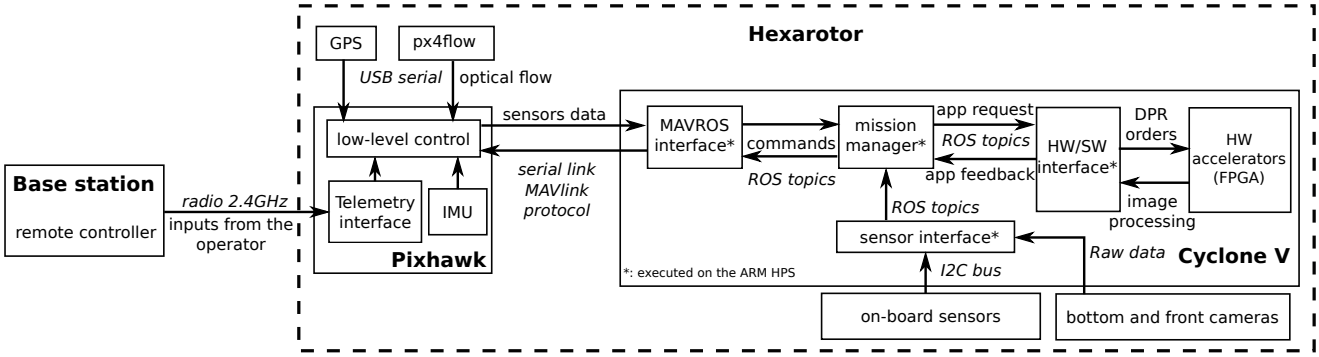

Fig. 4. Scheme of the UAV software setup.

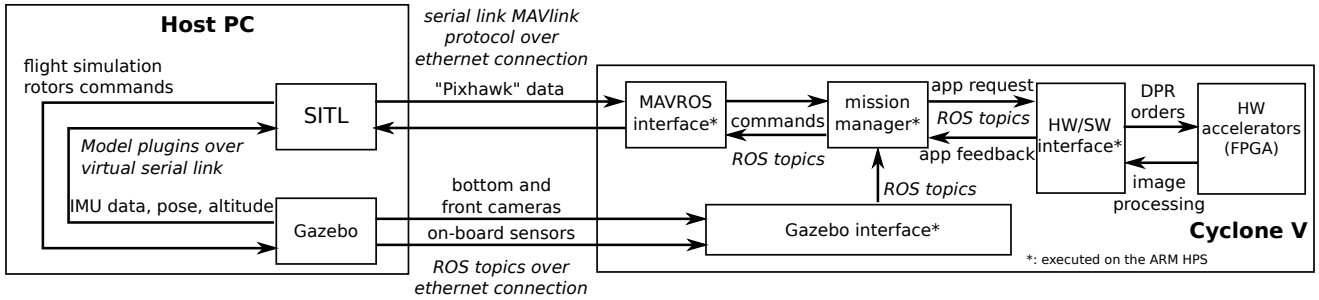

Fig. 5. Scheme of the HIL simulations software setup.

Correlation Coefficient (NCC). The normalization in NCC overcomes the problems of changing brightness. The scale is the median of all relative displacements of reliable points.

2) Detection: Detection is executed at different scales and positions. It performs object detection and recognition based on a learned object model. This model is composed of Binary Robust Independent Elementary Features (BRIEF) probabilities and patches representing the different object appearances. The first stage discards the windows with small variance, those that are likely to be background. After that, a BRIEF-based classifier [2] detects the windows where texture looks like those of the desired object. In the BRIEF classifier, 10 BRIEF codes of 13 different pairs of pixel are computed for each window already passed the first stage. To each resulting BRIEF code is associated a posterior probability measuring the probability of being the object of interest given the correlation result. If the mean of probabilities for the 10 BRIEF codes is lower than 50%, the corresponding window is rejected. For non-rejected windows, a similarity with respect to all object model patches is measured by computing the NCC. Finally, if the similarity is beyond a threshold, the corresponding window is considered as a detected object and the BRIEF probabilities are updated positively. Otherwise, the probabilities are updated negatively. A final stage aims to minimize the detected windows by clustering them in groups of overlapping windows.

B. *Emergency landing*

The emergency landing application seeks areas in the picture where the UAV can land. This is characterized by an area with a low luminance variance that indicates a low amplitude relief. A median filter is used on the grayscale image as a preprocessing step to remove noise. Then, a Canny filter is applied to detect area edges. A morphological closing is performed with dilation and erosion operations. If the area contrast is under a defined threshold the area is filled in white, otherwise the area is colored in black. The next step is to label each white areas that have a minimum size according to the UAV height to allow the landing. Thereafter, each white area selected is sorted such as the first area is the closest to the image center (location of the UAV).

C. *Simulation results*

First Table. I shows implementation results in terms of hardware resources for each application that can be implemented on each reconfigurable region. The percentage indicates the use ratio of a region according to the application. We observe that the detection application is the largest one and requires more than 90% of all ALMs and DSPs of a region. On the other hand, the emergency landing is only using 21% of ALMs.

In order to validate our all HIL platform with DPR, the simulation scenarios, namely the mission, were defined statically with predefined trajectories and application priorities assigned to the map regions. Detection, Tracking and Emergency Landing applications have been successfully executed in software or hardware according to the location, while basic UAV applications such as the navigation, obstacle detection and avoidance were running in software only.

Figure. 6 illustrates a phase of the HIL demonstration where the Emergency landing is running on one region. Part B is a view of the UAV downward facing camera, Part A shows the camera image after processing executed in hardware, Part C shows Gazebo 3D scene with the UAV in the center and part D shows a terminal with software and hardware timings of the emergency landing. Performance

results are summerized in Table II. The execution time of the software version of emergency landing is on average 300 ms against 8 ms for the hardware one. It generates a speed factor of 37.5 and relieves significantly the CPU load. Similarly, the execution of the software version of TLD requires 285 ms against 8.1 ms for the detection and 7 ms for the tracking both in hardware. We have considered the case where the detection and tracking are running in parallel on the two regions, this configuration brings a speed factor of 35. Nevertheless, the reconfiguration time requires 7 ms, which is of the same order of a hardware execution, so the reconfiguration rate must be limited. In the context of UAV mission, the evolution of application priorities and so the configuration rate is expected to be slow.

Finally, this case study provides a simple way to validate different sequences of configurations of the DPR architecture within a HIL simulation. Indeed, it allows designers to check the hardware behavior in a simulation near of real conditions. Moreover, we can perform an entire mission with different scenarios in order to limit risks for the UAV and its environment by anticipating possible events related to obstacles, target behaviour and UAV or Sensors failures. However the HIL simulation introduces a bias the designers must be aware of. Due to the ethernet connection, the UAV board needs to copy the camera image from Gazebo (the host PC). This overhead copy takes on average 7 ms, as indicated in part D of Figure. 6. This time must be subtracted from the total execution time in order to estimate the performances of the final embedded implementation. Finally, our work is by definition meant to be shared with the communities of robotics and embedded systems designers. Our open-source framework is available on github server which is still in work in progress[1].

TABLE I
IMPLEMENTATION RESULTS IN TERMS OF HARDWARE RESOURCES

	ALMs	Registers	DSPs	RAM(bits)
Tracking	6569 (66%)	6203 (16%)	34 (90%)	232203 (15%)
Detection	9534 (95%)	11124 (28%)	34 (90%)	581976 (37%)
E Landing	2125 (21%)	4042 (10%)	3 (8%)	128592 (8%)

TABLE II
PERFORMANCE RESULTS OF APPLICATIONS IN SOFTWARE AND HARDWARE VERSIONS

	SW execution time	HW execution time	Speed factor
Tracking	285 ms[a]	7 ms	40.7
Detection	285 ms[a]	8.1 ms	35
E Landing	300 ms	8 ms	37.5

[a] This execution time includes both SW Detection and Tracking parts using OpenTLD application.

V. CONCLUSION

In this paper we presented the first complete framework for HIL simulation that targets DPR architectures. This framework is composed of well known and open-source tools such as Gazebo, ROS and SITL. The simulation system has been tested on an Cyclone V FPGA board. It allows to specify and simulate a UAV mission in a virtual world but running real applications on the final embedded system. This systems is based on a DPR architecture that can be dynamically reconfigured during the mission. So UAV missions can be now tested with an embedded system that can benefit from DPR architectures in order to speedup high priority applications according to the scenario phases. The platform has been validated with real life computer vision applications executed with different HW and SW versions and dynamically configured on the FPGA during the simulation.

Our future work will be dedicated to the implementation of the advanced version of the mission manager. It will allows the UAV to fly autonomously and so to reconfigure the DPR architecture according to the mission scenario in an uncertain environment.

[1]https://github.com/Kamiwan/HPeC-sources

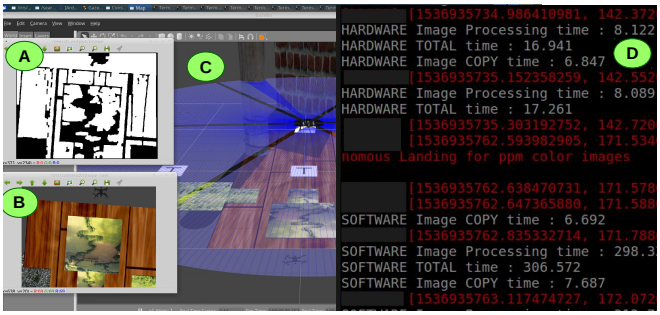

Fig. 6. Hardware In The Loop demonstration picture using DPR. A: image result from emergency landing image processing. B: View of the UAV's downward facing camera. C: Gazebo's world view. D: Terminal showing hardware and software timings of emergency landing application.

REFERENCES

[1] A Babushkin. Jmavsim, 2018.

[2] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV*, 2010.

[3] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. RotorS A Modular Gazebo MAV Simulator Framework. In *Robot Operating System (ROS)*, Studies in Computational Intelligence, pages 595–625. Springer, Cham, 2016.

[4] N. Gageik, P. Benz, and S. Montenegro. Obstacle detection and collision avoidance for a uav with complementary low-cost sensors. *IEEE Access*, 3:599–609, 2015.

[5] D. Göhringer, M. Hübner, E. N. Zeutebouo, and J. Becker. Operating system for runtime reconfigurable multiprocessor systems. *Int. Journal of Reconfigurable Computing*, 2011:3, 2011.

[6] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, R. Torrego, and T. Arslan. Microkernel architecture and hardware abstraction layer of a reliable reconfigurable real-time operating system (r3tos). *ACM Trans. on Reconfigurable Technology and Systems (TRETS)*, 8(1):5, 2015.

[7] A. Jara-Berrocal and A. Gordon-Ross. Scores: A scalable and parametric streams-based communication architecture for modular reconfigurable systems. In *DATE Conf.*, 2009.

[8] A. Jara-Berrocal and A. Gordon-Ross. An integrated development toolset and implementation methodology for partially reconfigurable system-on-chips. In *22nd ASAP Conf.*, 2011.

[9] D. Jung and P. Tsiotras. Modeling and hardware-in-the-loop simulation for a small unmanned aerial vehicle. In *AIAA Infotech@ Aerospace Conference and Exhibit*, 2007.

[10] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. In *20th Int. conf. on Pattern recognition (ICPR)*, 2010.

[11] L. Kessal, N. Abel, and D. Demigny. Real-time image processing with dynamically reconfigurable architecture. *Real-Time Imaging*, 9(5):297–313, 2003.

[12] N.P. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IROS*, volume 4, pages 2149–2154. Citeseer, 2004.

[13] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda. The erlangen slot machine: A dynamically reconfigurable fpga-based computer. *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, 47(1):15–31, 2007.

[14] K-D. Nguyen and C. Ha. Development of hardware-in-the-loop simulation based on gazebo and pixhawk for unmanned aerial vehicles. *Int. Journal of Aeronautical and Space Sciences*, 19(1):238–249, Mar 2018.

[15] M. Odelga, P. Stegagno, H. H. Blthoff, and A. Ahmad. A setup for multi-uav hardware-in-the-loop simulations. In *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 204–210, Nov 2015.

[16] T. Ohkawa, K. Yamashina, T. Matsumoto, K. Ootsu, and T. Yokota. Architecture exploration of intelligent robot system using ros-compliant fpga component. In *27th Int. Symp. on Rapid System Prototyping (RSP)*, 2016.

[17] S. Shah, D. Dey, C. Lovett, and A. Kapoor. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*, Springer Proceedings in Advanced Robotics. Springer, Cham, 2018.

[18] ArduPilot Dev Team. SITL simulator (software in the loop). `http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html`, 2016. [Online; accessed 13-July-2018].

[19] K. Yamashina, H. Kimura, T. Ohkawa, K. Ootsu, and T. Yokota. crecomp: Automated design tool for ros-compliant fpga component. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 138–145, Sept 2016.

[20] K.I. Yamashina, T. Ohkawa, K. Ootsu, and T. Yokota. Proposal of ros-compliant fpga component for low-power robotic systems. In *2nd International Workshop onFPGAs for Software Programmers(FSP 2015)*, Sep 2015.

[21] J. Yang, L. Yan, L. Ju, Y. Wen, S. Zhang, and T. Chen. Homogeneous noc-based fpga: The foundation for virtual fpga. In *10th Int. Conf. on Computer and Information Technology (CIT)*, 2010.