Multilevel host-compiled simulation framework for ROS-based UAV services using ArduCopter

Javier Merino
Dpt. TEISA
University of Cantabria
Santander, Spain
javierm@teisa.unican.es

Raul Gomez
Dpt. TEISA
University of Cantabria
Santander, Spain
raulgv@teisa.unican.es

Hector Posadas
Dpt. TEISA
University of Cantabria
Santander, Spain
posadash@teisa.unican.es

Eugenio Villar
Dpt. TEISA
University of Cantabria
Santander, Spain
villar@teisa.unican.es

***Abstract*— New services and business models based on drones are continuously being proposed. System engineering for these services have to include simulation as the cost of detecting design mistakes during the first prototype flights may be very high. If this fact is true in any complex system, in the case of drone-based services design mistakes may compromise the drones and the payload they carry increasing the associated cost. System verification has to be made at different abstraction levels so that each design step is verified. As a consequence, a multilevel simulation framework is needed. In this paper, such a simulation framework is proposed able to verify the system functionality and performance along the design process.**

Keywords—UAV, drone, ArduCopter, simulation, ROS, multilevel, host-compiled

I. INTRODUCTION

The number and importance of new services and business models based on drones is continuously growing. If in 2019 the Unmanned Aerial Vehicles (UAV) market was estimated at USD 19.3 billion it is projected to reach USD 45.8 billion by 2025, at a CAGR of 15.5% from 2019 to 2025 [1]. If currently, drones are applied in a reduced number of niches such as personal amusement, races, photography, public spectacles and police surveillance, once the regulatory frame is fixed, the number of potential applications will explode. Most of these new services will go beyond a single drone handled by a pilot. They will be based on smart, autonomous drones able to follow a mission and react satisfactorily to unexpected events. In many cases, the service will require an intelligent cooperation among several, even many drones in a swarm. In all these cases, the complexity of the functionality supporting the service may be exceptionally large, distributed on many devices of different nature such as drones, control stations, edge devices, servers and, in some cases, the cloud. Modeling the complete service, selecting the best functional distribution based on an analysis of performance, validating and verifying the solution and, finally, deploying it, is a complex task.

The drone itself is a mechatronic device in which the SW runs on a HW platform inside a mechanical structure allowing it to fly. This is possible because there is an autopilot electronics controlling the speed of each rotor accordingly to the orders coming from the controlling entity (either a radio-receptor receiving the orders from a pilot or a Ground Control Station (GCS), an internal plan or a smart SW on-board) and the signals provided by the sensors (gyroscope, accelerometer, magnetometer, GPS, etc.). Each drone flies in a physical world under mechanical and aero-dynamical laws ruling its movement and its interaction with the environment. Therefore, in the most general case, design and implementation of UAV-based services is a hard, time consuming and error prone Cyber-Physical System (CPS) design process. Detecting design errors once the service is deployed, even before commercialization, is very costly due to the following reasons:

1. Setting-up a real verification scenario for a service based on drones is costly and time consuming.
2. Detecting and locating errors during real operation is difficult.
3. Correcting the fault may require, in many cases, backtracking to earlier stages of the design process with the corresponding re-engineering costs and time.

As the verification scenario implies the use of real drones fully equipped with the payload required by the service, any mistake may compromise the drones and their payload.

UAV simulators have two fundamental pieces:

1. The model of the autopilot, which is the SW that runs on the drone receiving the commands from the pilot and the signals from the accelerometer, the gyroscope, and the magnetometer and generating, accordingly, the signals to the rotors.
2. The model of the physical environment of the drone, that is, the aerodynamics defining the relation among the signals applied by the autopilot to the rotors, their speed, and the movement of the drone in the air under concrete conditions. The resolution of the physical equations provides the next position, direction, and acceleration required by the sensors in the drone. This functional piece defines the test-bench exercising the autopilot functionality under a concrete scenario.

An additional component in many simulation frameworks is a real-time 3D engine able to feature rendering, dynamic physics and effects so that the drone flight can be visualized in a realistic scenario.

This work has been funded by the EU and the Spanish MICINN/AEI through the ECSEL Comp4Drones and the TEC2017-86722-C4-3-R PLATINO projects.

To minimize the number of design errors to be detected during deployment in field, simulation is widely used during the design process of a new drone [2] or the autopilot SW of the drone [3]. Several commercial and open-source simulators are available enabling to simulate a mission to be executed by previously available drones [4-5].

Smart drones can take autonomous decisions instead of depending continuously on the commands from the pilot or the GCS. They have to implement an additional functionality to be executed by the HW platform of the drone. Some services require the cooperation among several UAVs in a swarm. In that case, additional functions are required in each drone and, in some cases, in the GCS. Depending on the service to be provided the complexity of the complete system may growth significantly.

The growing use of drones and robots in general, foster the use of common standards. So, the Robot Operating System (ROS) is an open-source set of software libraries and tools widely used to facilitate the development of robot applications becoming a 'de-facto' standard. From drivers to state-of-the-art algorithms, ROS provides the required facilities for robotics projects [6]. Most applications in this domain use ROS. The C++ version of ROS is called ROScpp. ROS acts as an underlying middleware providing a series of communication features among nodes. Although the main Model of Communication is the publisher-subscriber, it also supports the client-server paradigm. Two separated parts can be identified in the ROS framework:

1. The functional part, that is, the user's application functionality developed in C++ using ROS facilities (ROScpp). This functional part may have as many ROScpp nodes as required,
2. The ROS core, which handles all the communication mechanisms among nodes, and is in charge of timing, synchronization, etc. The ROS core is an underlying code which the simulation framework has to take into account but which is transparent for the system engineer.

ROS has built-in time and duration primitive types, in order to specify a specific moment or a period of time, respectively. Although ROS is meant to work with wall-clock (Operating System) time, it is possible to set-up a simulated clock.

Most drones use MAVLink for drone communication [7]. MAVLink is a lightweight messaging protocol for communicating with drones and between onboard drone components. ROS includes MAVROS, a communication driver for various autopilots with MAVLink communication protocol. Additionally, it provides UDP MAVLink bridge for ground control stations [8].

Any modeling and simulation environment for drones should support ROS models. As in many cases the interface with the autopilot is through MAVLink, the simulation environment should be able to integrate it.

The SESAR Joint Undertaking has recently drafted the U-Space, a set of new services relying on a high level of digitalization and automation of functions and specific procedures designed to support safe, efficient and secure access to the European airspace for large numbers of drones [9]. Services complying with the U-Space will have to implement a large set of procedures closely interacting among them. Modeling, simulation, performance analysis, validation and verification of such complex services will require of adequate system design frameworks where the system engineers can model, analyze, and implement the service ensuring its correct behavior once deployed in the field.

Therefore, drone-based applications require long and complex design processes, as they must deal with strict non-functional requirements such as criticality, timeliness, reliability and safety. The huge number of simulations, performance analysis and evaluations to be performed requires powerful simulations technologies combining high simulation speed and accuracy. Moreover, simulation-based verification should be performed all along the design process. This requires multi-level simulation capability with two main purposes. On the one hand, to enable adding and simulating new details as they are decided. In an initial, functional simulation one pretends to validate the behavior of the system under a certain test-bench. Once the functional components are mapped to concrete HW devices, one pretends to estimate non-functional constraints such as execution time and energy. The model of the drone(s) can be as simple as a moving object at constant speed or as detailed as a full mechatronic model. On the other hand, one may require to analyze in detail a certain component (i.e. a drone in a swarm) inside a functional model of the rest of the system

Host-compiled simulation is a powerful, flexible approach able to achieve fast, performance simulation of software running on complex embedded systems [10]. The background technology used in this work is VIPPE, a host-compiled simulation tool able to support simulation and performance analysis of complex systems as part of the S3D Model-Driven system-design framework [11].

In this paper, a multi-level simulation framework is proposed based on VIPPE, which has been extended to simulate system models using ROS and to integrate drone models at different levels of accuracy vs speed, from a simple functional model to a model integrating the autopilot (i.e. Ardupilot) and the physical environment (SITL or Gazebo).

II. Related Work

As commented above, simulation is essential when developing UAV-based services as design errors should be detected before deployment. Matlab/Simulink has been used to develop ad-hoc simulators both when designing a new drone [2] or only the autopilot SW of the drone [3]. In both cases, it is necessary to model the behavior of the autopilot SW and the physical equations modeling the aerodynamics of the drone. Being a Simulink model may limit its simulation speed and, therefore, its scalability. MATLAB includes a ROS Toolbox [12]. The toolbox provides an interface that connects MATLAB and Simulink with ROS, allowing you to create a network of ROS nodes. The toolbox allows to verify ROS nodes through desktop simulation and by connecting to external robot simulators such as Gazebo. Nevertheless, MATLAB/Simulink is not an appropriate framework for simulation and performance estimation at several abstraction levels.

In order to improve the simulation speed, several flight simulators have been proposed covering UAVs. The fundamental pieces they include are the autopilot and the model of the flying aerodynamics. In many cases, a 3D virtual

reality engine is provided [4-5]. Each one of these added features slow-down the simulation speed significantly. Most of these simulators are 'Real-Time' (RT), that is, simulated and simulation times are the same (1/1) so that there is possible a direct interaction between the executable model and a human-in-the-loop as a pilot or a GCS operator. Nevertheless, RT simulation makes difficult the simulation of large services (i.e. a full day of a parcel delivery service).

An important conclusion of these analysis about drone simulation is that currently, it is no longer necessary to build an UAV autopilot and flight simulator from scratch. This makes sense only when the service to be modeled is complex enough and there is no need to analyze all the details of its behavior. The modeling and simulation of UAV swarms configuration and mission planning falls into this case, justifying the development of an ad-hoc simulation framework [13]. In this case, the realistic but slow physics simulation is substituted by a simple model of the drone movement. This brings a pure functional simulation of the SW controlling each drone in the swarm. Parallelization should improve simulation speed but it is not always an easy task [14]. Functional simulation brings a way to verify the SW in complex drone-based services [15].

Functional simulation can incorporate estimations of extra-functional characteristics such as execution times and energy. When a more accurate estimation is required, host-compiled simulation or virtualization can be used.

III. Problem Description

From the analysis of the state-of-the-art, one concludes that there is a lack of multi-level simulation environments where a complex, drone-based service can be analyzed holistically but with different levels of details under the same simulation test-bench. In order to improve scalability, the simulation should be as fast as possible (AFAP), in addition to Real-Time (RT) so that the simulated time can be as higher as possible than the simulation time (i.e. being able to simulate a full day of a parcel delivery service in minutes). This is a first problem to be solved as both the ROS infrastructure as the drone simulators are RT.

The first, more abstract simulation model will be pure functional. Depending on the most appropriate ratio accuracy/speed, the drone-based service SW can be simulated without any performance estimation, with workload modeling (constant execution times and energies) or using host-compiled simulation. This SW will include ROScpp code to control the drones. The ROS infrastructure will be simulated with a simple ROScpp test bench able to react in the same way as the real drone would do. Nevertheless, both the drone autopilot and the physics simulator are substituted with a simple drone movement model with constant vertical and horizontal speeds. This simulation model is targeted to the verification and debugging of the application SW.

The next level will be adding the ROS infrastructure. This would allow to detect any mistake in the interaction between the functional SW and the ROS implementation of the communication mechanisms among components without including the details of the drone autopilot and aerodynamics.

In a third simulation level, the autopilot and the physics simulator are added for a more realistic simulation and detailed timing analysis. This will slow-down the simulation but will allow detecting potential faulty interactions among the application SW, the ROS infrastructure and the drone behavior. An additional problem to be solved is the integration of the ROS time and duration primitives within the system simulation environment. In the same way, it is necessary to integrate the timing of the drone simulator into the environment keeping the correct behavior of the physical model.

The generation of the different simulation environments commented above should be simple. Ideally, generated automatically from the same model.

IV. Modeling Framework

In order to better explain the simulation framework proposed, it is important to describe the modeling methodology followed to describe drone-based services. Figure 1 shows the fundamental elements of a model including ROScpp components. The model is divided in two parts, the system model, which is invariant, and the test-bench. The system model is composed by functional components in C++ and components controlling the drones in ROScpp. The C++ components communicate and synchronize among them, with the ROScpp components and with the test-bench through Required-Provided (R-P) services.

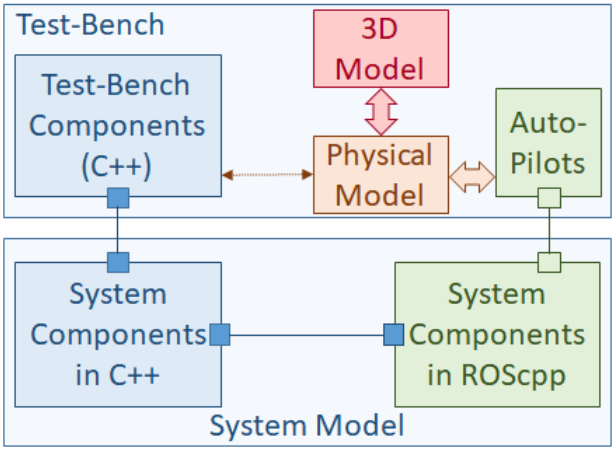

Figure 1: Elements of a S3D model of a drone-based service.

The ROScpp components communicate with the C++ components using R-P services but they may use ROS function calls among them. These function calls will be implemented using ROS communication mechanisms based on the Publish-Subscriber paradigm supported by the ROS infrastructure. Each ROS component controlling a drone will be connected with the drone's autopilot. This communication will use also ROS function calls. The autopilot for the drone is considered part of the test-bench. The autopilot has to be connected to the model of the physical environment simulating the drone aerodynamics. As commented above, the physical model may be connected to a 3D graphics engine of the environment inside which the drone moves. This 3D environment may be shown to the user either as a subjective view from the drone or as a third-party observer. The main use of the 3D graphical output is as a Graphical User's Interface (GUI). In some cases, the physical model has to interact with the system through the C++ test-bench (i.e. an alarm when the drone deposit a good in a reception box). As we are interested in AFAP simulation, the 3D graphic engine will not be integrated in the framework.

Although the code associated to the system components is invariant it will be modeled at different abstraction levels

along the design process thus requiring of a multi-level simulation framework.

V. Multilevel Simulation Framework

In this section, the simulation framework will be described. Then its use in multi-level simulation will be explained.

A. Simulation Framework

ArduCopter has been chosen as the multicopter autopilot platform for this work. ArduCopter makes use of the MAVLink serial protocol for control and telemetry communication. MAVLink will make use of a radio link when the communication source is outside the drone (the GCS or a pilot). Internal communication makes use of a serial port. Although it is widely used in this field and supports many different features, its usage is not trivial for a software developer and requires a deep knowledge of the protocol.

As a consequence, the integration of ArduCopter requires an underlying MAVROS node, working as a wrapper in charge of the translation between ROS and MAVLink messages, thus avoiding to program directly using MAVLink. This additional code will run on the same HW device to which the corresponding ROScpp node has been mapped. Usually, the GCS controlling the drone. In the case of smart drones, the ROScpp code and the corresponding MAVROS wrapper will be executed by an additional board coupled on the drone and connected via serial port to the autopilot board. This configuration allows moving the drone communication interface from MAVLink to ROS and treat the drone as a ROS node.

ArduCopter makes use of SITL (Software in the Loop) simulator. This software emulates the drone aerodynamics inside a realistic physical environment. Basically, SITL takes as input the Pulse Wide Modulation signals to the rotors and generates the signals to the drone sensors (i.e. acceleration and direction) periodically, depending on the previous speed and direction. Non ideal conditions as gusty wind can be taken into account. SITL can be substituted by other physical environment simulators such as Gazebo [16].

In its most general form, the architecture of the simulation framework is shown in Figure 2:

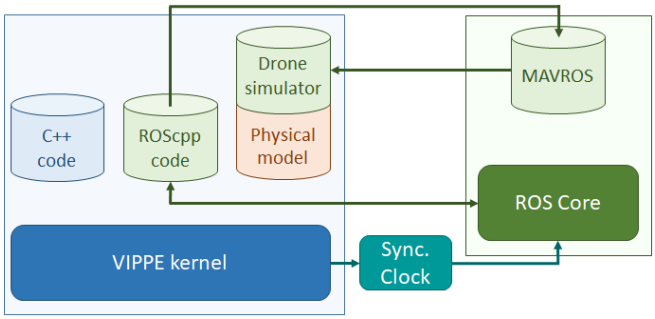

Figure 2: Multilevel drone simulation framework.

The simulation framework requires two interrelated simulation domains. On the one hand, the functionality under the control of the VIPPE kernel. It is in charge of the simulation of the C++ code, the ROScpp code and the drone simulation code. The drone simulation code includes the model of the autopilot and the model of the physical aerodynamics. On the other hand, the ROS functionality which includes the ROS Core and the MAVROS in charge of the translation between ROS and MAVLink supporting the communication with the drone. By default, the ROS timing is related with the OS (i.e. Linux) clock. Nevertheless, ROS supports using simulated time instead of wall-clock time by setting the */use_sim_time* configuration parameter to true. In order to coordinate the two domains, a ROS node is created for clock synchronization. It obtains VIPPE current simulated time every kernel loop and publish it to the */clock* topic of ROS, forcing it to advance its time at the VIPPE rate.

B. Multilevel Simulation

Once the simulation framework is described, the different simulation levels are now introduced. The multilevel approach aims to facilitate the development of the application, allowing the possibility of testing it on a simple, pure functional model of a drone, and once its correctness is verified, easily migrate to a more detailed and accurate model integrating ROS and the drone simulator. In our case, ArduPilot and the physical model (SITL or Gazebo).

Three different levels are considered. The first level is based on a simple drone movement model with constant speeds, so its movements and elapsed time can be easily calculated, allowing a fast simulation. This level does not make any use of the ROS facilities, so neither the ROS core, nor the MAVROS, nor the clock synchronization elements are needed. The elements of the simulation framework used at this level are those shown in Figure 3:

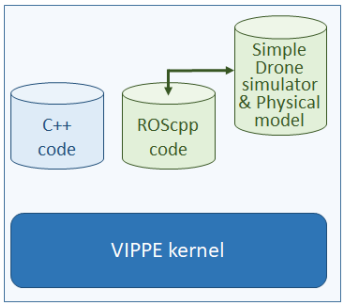

Figure 3: Functional simulation elements.

The ROS function calls made by the ROScpp components are executed as direct orders to the drone by the drone simulator. At this abstraction level, the ROSCpp and the C++ code are usually handled without execution times. That is, the code is supposed to be executed in no-time. The advantage using VIPPE comes from the possibility to use workload models when the code has not been developed yet and only rough estimations of its performance can be made. This enables to introduce estimated execution times and energy consumption to each function. Once the full code is available, host-compiled simulation technology would provide much more accurate results. It is worth mentioning that, from the user's perspective, moving from one abstraction level to the other only affects the model of the drone as the VIPPE simulation technology for the C++ and ROScpp components is multilevel by itself. The model of the drone can be an object that switch-on and moves vertically and horizontally at constant speeds with an aerodynamics as simple as:

$$\Delta e_{(x,y,z)} = v_{(x,y,z)} \Delta t$$

that is, the space moved by the drone in any direction is equal to the component of the speed of the drone in that direction multiplied by the time elapsed by the movement.

The second level keeps the simple model for the drone but integrates the ROS core. In this way, it is possible to verify the complete functionality including the behavior of the ROS infrastructure. At this level it makes more sense to use a timed model for the C++ and ROScpp components as both simulation domains are synchronized by the VIPPE time. Thus the temporal behavior can be analyzed and its interaction with the ROS infrastructure taken into account. The drone is still modeled using the simple functionality commented above but now, as an actual ROS node. The corresponding instantiation of the simulation framework is shown in Figure 4.

The last, more detailed simulation set-up introduces all the previously described ROS infrastructure together with the autopilot drone model (i.e. ArduCopter), as shown in the general framework in Figure 2. Now, MAVROS is needed as the communication with the autopilot usus MAVLink. In this level, two physical models are tested, showing the flexibility of the proposed simulation framework to integrate third-party tools. In the first configuration, ArduCopter uses SITL as physical environment simulator. On the second one, Gazebo simulator is attached to be used as the physical simulator (some SITL interfaces remain to simulate the I/O devices of the drone). Introducing Gazebo is relevant since it presents a wide series of features such as multi-vehicle deployment and collision simulation, so external actors can be considered during the application development. Furthermore, simulation speed can be configured for a faster simulation. However, if this speedup is increased excessively, the host processor could not be fast enough to perform all required calculations and result on a poor performance of the drone.

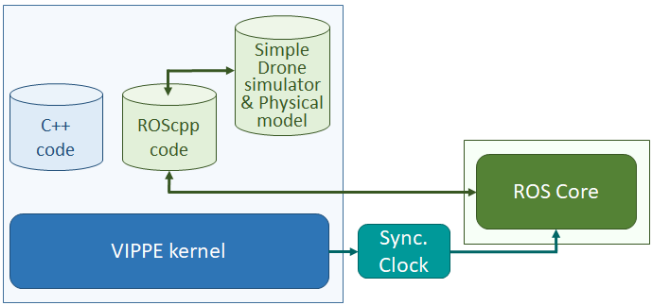

Figure 4: Functional simulation including ROS.

VI. Experimental Set-up and Results

In order to validate the multilevel simulation framework proposed, it is applied to a representative use-case, a drone-based delivery service of goods. The service and its environment are shown in Figure 5:

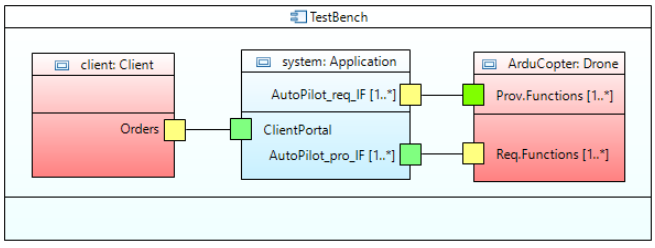

Figure 5: Goods delivery service in its test-bench.

The system has two interaction points. On the one hand, the service portal through which the clients access the service. On the other, the communication links with the drone fleet sending and receiving the ROS commands and data to and from each one of the drones. The functional architecture of the system is shown in Figure 6.

The 'Delivery_Central' is the functional component storing the data-base of goods. Through the 'ClientPortal the customers can find the products that the company sells and make a selection of them. The client has to register to the service providing the identity, payment method and address. After paying the price, the order is accepted and the delivery of the products starts. The first action is to ask the 'Route_Generator' to find the most appropriate flying path to the client's address satisfying all the U-Space regulations. The path is sent to the 'Drone_Controller' component. The 'Drone_Controller' selects a drone in the fleet and is sends the flying path to it. Each time a waypoint is acceded, the drone informs the 'Drone_Controller' until the mission is finished. If something goes wrong the 'Drone_Controller' is advised and the information sent to the 'Delivery_Central'.

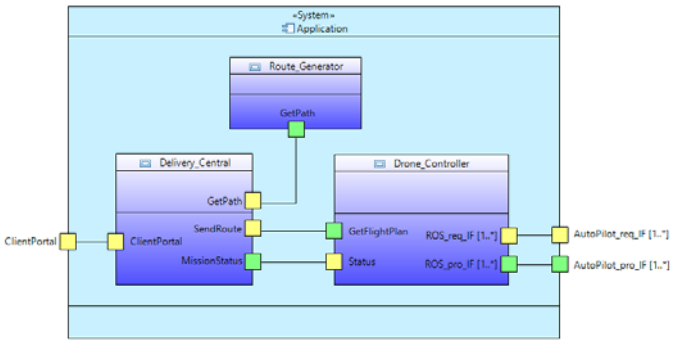

Figure 6: Functional architecture of the goods delivery service.

Simulation results are presented in terms of the usage of CPU by the different simulation elements obtained as the proportion of the total CPU time used by the whole simulation framework. Experiments have been made using an Intel Core i5 3470 at 3.2 GHz. Results obtained are shown in Table 1:

Table 1: Simulation results

	% CPU						sec/sec
Sim. Type	App. Code	VIPPE Kernel	Syn. Clock	ROS Core	MAVROS	Drone Sim.	Sim. Speed
Workload model	21.62	78.38	-	-	-	*	660
Workload + ROS	15.15	37.88	4.55	42.42	-	*	27.23
ArduCopter SITL	3.85	21.67	6.62	3.57	5.74	58.55	2.29
ArduCopter Gazebo	2.18	12.35	3.97	2.06	3.21	76.23	1.65

For the functional simulation, only the application code and the VIPPE kernel are considered since there are no ROS elements and the simple model of the drone is an additional C++ code so that it is included in the VIPPE CPU usage, as part of the test-bench. This has been highlighted in Table 1 with an asterisk (*). As the focus of the paper is the simulation technology, the associated C++ code for the 'Delivery_Central' and the 'Route_Generator' is not fully developed so that a workload model is used. As shown, this configuration provides a very fast simulation of the service, obtaining a simulation speed of 660 s/s. Since the total simulation time is very low, VIPPE kernel utilizes most part of this time. This result shows that it is possible a functional simulation of very complex services based on drones when pure functional models of the drones are used. The focus, can be put in assuring that the functional and extra-functional constraints can be satisfied by using this functional

architecture. Extra-functional constraints can be analysed on workload models. In a workload model a constant execution time and energy consumption is associated to each time each function is executed. They can be mean values or worst-case estimations for each component.

The next step would be to add the ROS infrastructure in order to detect any faulty behavior caused by an incorrect interaction between the C++ code and the ROScpp code when the synchronization and communication services offered by ROS are executed actually by the ROS infrastructure. As showed in Table 1, the ROS infrastructure has a deep impact in performance as the simulation speed is reduced to 27.23 s/s. Nevertheless, this is still high enough to enable the simulation, verification and performance analysis of large systems. The synchronization clock execution time is kept small. The contribution of the VIPPE kernel is still significant.

The last level analyzed is when a detailed model of the autopilot and the drone aerodynamics are used. The autopilot selected is ArduPilot working with two different physical simulators, SITL, the native one and Gazebo, an external one to which an interface exists. As derived from the results in Table 1, when just a drone and its aerodynamics is modeled in detail, the simulation speed is strongly reduced. In the case of ArduCopter-SITL, the percentage of CPU consumed by the models of the autopilot and the physical environment is the largest (58.55%). The execution time required by MAVROS is kept small. The simulation speed is reduced to only 2,29 s/s, still enough to support RT simulation or even, its usage as a digital twin. Gazebo is even heavier with a total CPU usage of 76.23% from which, 35.67% is the autopilot load and 40.56% the Gazebo physical model. The simulation speed is reduced to 1.65 s/s, still higher that RT.

VII. CONCLUSIONS

In this paper, the required simulation infrastructure allowing the verification and performance analysis of drone-based applications along the design process has been set-up. Different models can be applied at different abstraction levels used along the design process of those systems. The infrastructure has shown to provide very fast simulation speed when only workload models for the drone and the application components are used so that it can be applied to complex services. Based on usual results found in literature, if instead of a workload model, a host-compiled simulation model is used, simulation speed would decrease around one order of magnitude leading to simulation speeds of around 70 s/s on a common desktop PC. This would allow to analyze in detail the execution time, delays and energy consumption of specific components to be optimized as part of the whole application. The introduction of the detailed drone autopilot and aerodynamics slow-down the simulation speed dramatically. This means that only a small number of drones might be simulated at a low level of abstraction while for the rest, a high-level model can still be used. The focus in this case would be put on the detailed behavior of the critical drones in the application.

The simulation infrastructure proposed puts the time advance under the control of the system simulator, in our case VIPPE.

The paper shows how this can be done for the ROS infrastructure, using a synchronization clock but also for the drone simulators integrating them into the simulation framework. As the simulation infrastructure is the same, the generation of the different simulation models is seamless for the system engineer using the tool.

REFERENCES

[1] Unmanned Aerial Vehicle (UAV) Market. MarketsandMarkets. 2020.

[2] K. Patel and J. Barve. Modeling, simulation and control study for the quad-copter UAV. 9th International Conference on Industrial and Information Systems (ICIIS), Gwalior, 2014, pp. 1-6, doi: 10.1109/ICIINFS.2014.7036590.

[3] P. Lu and Q. Geng. Real-time simulation system for UAV based on Matlab/Simulink. IEEE 2nd International Conference on Computing, Control and Industrial Engineering, Wuhan, 2011, pp. 399-404, doi: 10.1109/CCIENG.2011.6008043.

[4] A. I. Hentati, L. Krichen, M. Fourati and L. C. Fourati. Simulation Tools, Environments and Frameworks for UAV Systems Performance Analysis. Proceedings of the 14th International Wireless Communications & Mobile Computing Conference (IWCMC), Limassol, 2018, pp. 1495-1500, doi: 10.1109/IWCMC.2018.8450505.

[5] E. Ebeid, M. Skriver, K. H. Terkildsen, K. Jensen, U. P. Schultz, A survey of Open-Source UAV flight controllers and flight simulators, Microprocessors and Microsystems, V.61, 2018, pp.11-20.

[6] Ros.org.

[7] https://mavlink.io/en/.

[8] http://wiki.ros.org/mavros.

[9] A. Prestigiacomo. U-SPACE Services Implementation Monitoring Report. European Organization for the Safety of Air Navigation (EUROCONTROL). 2020.

[10] O. Bringmann, W. Ecker, A. Gerstlauer, et al., “The Next Generation of Virtual Prototyping: Ultra-fast Yet Accurate Simulation of HW/SW Systems”, Proc. of DATE 2015.

[11] F. Herrera, J. Medina, E. Villar: "Modeling Hardware/Software Embedded Systems with UML/MARTE: A Single-Source Design approach", in Soonhoi Ha and Jürgen Teich (Eds): "Handbook of Hardware/Software Codesign", Springer. 2017.

[12] https://es.mathworks.com/products/ros.html.

[13] Yi Wei, M. Brian Blake, Gregory R. Madey. An Operation-Time Simulation Framework for UAV Swarm Configuration and Mission Planning. Procedia Computer Science. Vol. 18,,2013, pp. 1949-1958. doi.org/10.1016/j.procs.2013.05.364.

[14] J. J. Corner and G. B. Lamont. Parallel simulation of UAV swarm scenarios. Proceedings of the Winter Simulation Conference, Washington, DC, USA, 2004, pp. 363, doi: 10.1109/WSC.2004.1371336. gazebosim.org.

[15] N. Pathak, S. Misra, A. Mukherjee, A. Roy and A. Y. Zomaya. UAV Virtualization for Enabling Heterogeneous and Persistent UAV-as-a-Service. in IEEE Transactions on Vehicular Technology. Vol. 69, no. 6, pp. 6731-6738, June 2020, doi: 10.1109/TVT.2020.2985913.

[16] http://gazebosim.org/.