Proc. or the 17th Int. Conf. on Autonomous Robot Systems and Competitions (ICARSC 2017), Coimbra, Portugal, April 26-28, 2017.

Hardware-In-The-Loop Development Framework for Multi-Vehicle Autonomous Systems

Luigi Pannocchi, Mauro Marinoni and Giorgio Buttazzo
Scuola Superiore Sant'Anna, Pisa, Italy
Email: {l.pannocchi, m.marinoni, g.buttazzo}@santannapisa.it

***Abstract*—In applications involving unmanned aerial vehicles, the use of simulation environments is typically employed to speed up the development phase, reduce the associated costs, and in particular to safely verify and validate the software behavior without jeopardizing the hardware in case of faults or bugs. In addition, testing other properties as scalability, robustness, and fault tolerance is much more convenient in simulation.**

Communities dealing with unmanned autonomous vehicles often develop their own custom simulation frameworks, which are often tailored to the specific system under development, with the consequence that they are either incomplete or not fully supported.

To fill such a gap, this work presents a hardware-in-the-loop development simulation framework for multi-vehicle autonomous systems, addressing in particular maintainability, predictability, and capability of integrating multiple heterogeneous autopilot boards and user interface applications. The proposed framework also supports a multi-agent configuration, which is a noteworthy novel characteristic, since existing frameworks often deal with a single agent.

I. Introduction

In recent years, unmanned aerial vehicles have been used in several application domains and many contributions have been proposed to improve their performance and autonomy. The aroused interest has mainly been driven by their flexibility and scalability, making them usable in a multi-agent framework in very different settings [1] [2]. The continuous improvements of computational platforms and materials paved the way towards real "autonomous" vehicles capable of interacting with dynamic environments in a wide range of circumstances [3] [4] [5]. Significant progress has also been made on miniaturizing hardware platforms for unmanned aerial vehicles (UAVs), as well as providing programming guidelines [6] to promote code reusability and maintainability.

In spite of the support provided by several research communities, the development of multi-agent systems still requires considerable effort when testing the software related to navigation, guidance, control, and collaborative tasks.

The possibility of validating and testing the software behavior on a realistic simulation framework would allow significant benefits, as reducing the development time, avoiding crashing the vehicles due to software failures, and evaluating the reaction to peculiar situations that could be too rare or not reproducible in practice. The possibility of evaluating control software in a simulation framework is particularly important in a research setting, where not fully tested open-source solutions are typically adopted to develop innovative approaches.

978-1-5090-6234-8/17/$31.00 ©2017 IEEE

Reproducing realistic scenarios in the simulation environment is a crucial objective in the development process, often achieved through hardware-in-the-loop simulations. This method, schematically illustrated in Figure 1, consists in running the control software directly on the board of the vehicle, which interacts with a virtual environment. In this setting, the physical sensors and actuators of the vehicle are disabled and replaced with virtual counterparts running in the simulator.

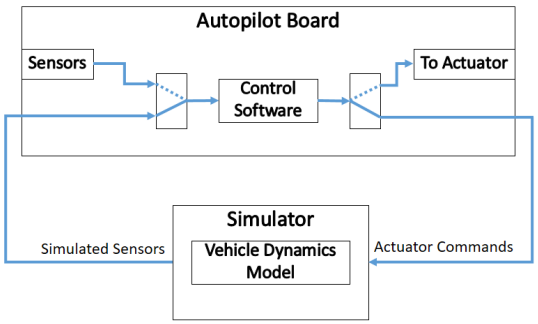

Fig. 1. Hardware-in-the-loop scheme.

Since the aim of the simulation process is also to test high-level functions (i.e., obstacle avoidance, target recognition and tracking, integration of video information for navigation, etc.), it is crucial to model all the interactions between the vehicle and the virtual environment, implementing proper physical models.

The success of a simulation tool also depends on its maintainability and interoperability with other software. It should be possible to use the tool with a large variety of systems, and any possible upgrade should be easy to do. Without such properties the tool is likely to become obsolete in a short time.

Research communities dealing with unmanned autonomous vehicles often develop their own custom simulation frameworks. As a consequence, such tools are often too specific and tailored to the system under development, and not always documented and supported.

To fill such a gap, this paper presents a hardware-in-the-loop simulation framework for multi-vehicle autonomous systems with the following characteristics:

1) Support for multi-agent scenario;

2) Modular design to improve maintainability;
3) Support for integrating generic autopilot boards;
4) Possibility to interoperate with a wide range of user interfaces for autonomous vehicles;
5) Support for testing high-level functions of the vehicles;
6) Precise time management to increase the realism of the simulation.

The rest of the paper is organized as follows. Section II presents a brief survey of the related literature. Section III introduces the proposed framework, its main components and the actual implementation used in this work. Section IV gives a detailed description of the software structure of the framework. Section V reports some experimental results aimed at testing the timing properties of the system. Finally, Section VI states our conclusions and future work.

II. Related Work

Many tools have been developed within the research community addressing different application scenarios. A valuable effort to classify existing simulators depending on their provided functionalities has been presented by Parodi et al. [7].

Focusing on hardware-in-the-loop simulators, several solutions have been proposed in the literature [21] [24] [9] [22] [1], but they all consider a single vehicle and do not extend to a multi-agent scenario. To manage multiple vehicles the simulator has to be properly designed to support such a feature. Kamali and Shikha [8] presented a simulation model developed in *Matlab/Simulink* and running on an FPGA to support real-time communication on I/O data buses (I2C and SPI). This approach allows achieving a good level of realism but it does not provide good scalability, because all the communication interfaces need to be wired to the FPGA that is in charge of simulating the models. A similar approach has been used by Pollini et al. [9], where *Matlab/Simulink* is used to simulate a single model on a PC, implementing a synchronization procedure between the autopilot board and the simulator.

The simulation of a vehicle can also be done using dedicated simulation tools, like *X-Plane*[1] or *FlightGear*[2], as done by Santos [10]. The major drawback of using a dedicated flight simulator, however, is that the overall execution rate is coupled with the refresh rate of the video. In addition, the user has not full control on the mathematical model. For this reason, Lugo-Cardenas et al. [11] implemented the model dynamics from scratch using an open-source simulator. Another useful feature of this solution, often missing in many existing tools, is the capability of embedding heterogeneous platforms in the simulator through the MAVLink protocol [12]. Using such a protocol, in fact, it is possible to connect all the supported autopilot boards without additional coding effort.

Unfortunately, there are only very few works supporting hardware-in-the-loop simulation for a multi-vehicle system. Often the multi-agent scenario is proposed within a fully simulated environment, with a very limited interaction between vehicles and the virtual environment, as in [13]. A hardware-in-the-loop simulation tool for multi-agent systems has been proposed by Kamal and Shumaker [14], who also considered gimbaled platforms. While such a solution is highly realistic, it has the problem of being quite expensive and difficult to scale with many vehicles.

Parodi et al. [7] proposed a well-structured simulation framework for underwater vehicles able to manage multiple heterogeneous agents. The simulator models the environment, including obstacles, water properties and constraints related to communication in water, but the framework does not include a 3D visualization. In addition, considering the possible problems related with communication between the different components, it would have been interesting to see some results about the timing properties of the system (latency, periodicity of messages, etc.).

Indeed, a proper characterization of the timing properties is quite important for a hardware-in-the-loop simulator, since the hardware should be provided with simulated sensor/actuator data as if they were produced by real devices. Providing simulated sensory data with high latency or jitter is like introducing a non constant artificial delay in the control loop, which would make the simulation less realistic. Such a timing characterization is rarely reported by authors, with very few exceptions. For instance, Pollini et al. [9] implemented a synchronization process between the autopilot board and the simulator, reporting the timing properties. Mueller [15] addressed such a problem by running the simulation code directly on the autopilot board, but this solution increases the workload of the autopilot processor and could alter the realism of the test.

III. System Description

The overall structure of the simulation framework consists of four main components and a message router, as illustrated in Figure 2.

The message router guarantees a correct interoperability among components and ensures that data exchange is carried out within bounded delay, also considering the worst-case blocking times on the access to shared resources. The communication with the ground station and the synthetic environment is performed using the UDP protocol, which allows spreading the components over separate machines to better distribute the computational workload.

1) *Autopilot boards*
   The aim of the simulation tool is to test the control software running on the target autopilot boards. When in a hardware-in-the-loop configuration, each board expects sensor data from the simulator and produces control signals back to the simulator. In addition, a board can exchange messages as during normal functioning, for example with a ground station. The system supports boards that can communicate via serial port or network interface using the *MAVLink* protocol. The rationale behind this choice is that *MAVLink* has become a de

[1] http://www.x-plane.com/ (X-Plane Flight Simulator)
[2] http://www.flightgear.org/ (FlightGear Flight Simulator)

facto standard communication protocol for open-source autopilot boards. In this way, the proposed simulation framework natively supports all of them without requiring ad-hoc modification.

2) *Simulator*
The simulator takes the actuation commands from each autopilot and uses the model of the vehicle dynamics to simulate the sensory data. In order to achieve good performance, scalability, and have full control on the model, the functions for describing the vehicle dynamics and the output of the sensors have been implemented using the C language. The simulation frequency can be changed by the user and it is not bounded by the video refresh rate, as in common flight simulators. The sensory data output frequency can be decoupled from the simulation frequency by simulation sub-stepping, that is, by sending the sensory data every N simulation cycles, where N is the number of sub-steps.
3) *Synthetic Environment*
This component provides the capability of 3D visualization of the vehicles in a synthetic environment. Visualizing the vehicles in a virtual environment is a valuable feature that simplifies the verification of the developed functions. It is also possible to simulate the output of cameras on the vehicles for testing advanced functions, as vision-based maneuvers. An important feature of this component is that it can be employed not only as a bare viewer, but also to retrieve information about the interaction of a vehicle with the synthetic environment, such as collisions with virtual obstacles.
4) *Ground station*
The ground station is the application that allows user interaction with the autopilot board through an user-friendly interface. It allows the user to change the values of autopilot firmware parameters and perform mission planning and flight control operations. It also provides an interface to visualize the telemetry data and the status of the connected vehicles. Instead of proposing an ad-hoc ground station, in line with the goal of modularity and interoperability, the proposed framework can operate with any ground station implementing the *MAVLink* protocol over UDP. This is the case with most of the common available ground station applications, which can be tested, together with the autopilot board, in this simulation environment.

A. Actual Implementation

The modeling of the simulated vehicles and its sensors has been carried out using *Matlab/Simulink*, taking advantage of C code generation to ensure scalability and fast execution, which is required in case of multi-vehicle systems. In this way a trade-off between maintainability, extensibility and scalability has been achieved. In this work the vehicles have been modeled as identical quadrotors, but it is possible to add also other kind of vehicles, simply modeling them with *Matlab* and generating the C code.

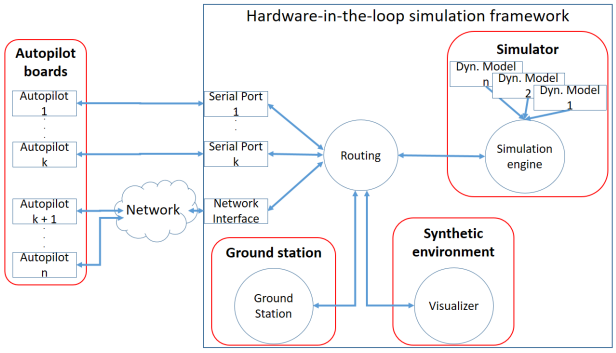

Fig. 2. Overall structure of the hardware-in-the-loop simulation framework.

The ground station used in our implementation is *QGroundControl*[3], that is a free open source application allowing full modification and code analysis. This application supports the *MAVLink* protocol for communicating with the connected vehicles. It supports different communication interfaces for connecting to the vehicles, also allowing the required UDP protocol. Through it the user can visualize the vehicle on a map (*Google Maps, Bing Map*), plan missions putting waypoints directly on the maps, plot telemetry data and simplifies several operations on the vehicles, such as flashing firmware, setting up the controller parameters and the remote controller.

The synthetic environment has been designed with *Unreal Engine*[4], which is a free complete suite of game development. It makes possible to design realistic scenarios without directly concerning about complex visual effects, which are managed automatically by the graphical engine of the application. This is necessary to support the design of computer vision solutions, where real common phenomena like light reflections, shadows, and mist constitute a problem that must be taken into account. The suite includes useful libraries to create realistic effects, move objects, measure the distance between points, manage events like collision, and so on.

IV. Hardware-in-the-loop Framework Structure

A generic hardware-in-the-loop simulation (see Figure 1) consists of two interacting elements: the software under test, running on the autopilot board, and the simulator application executing on a separate machine, respectively. Since the two elements run on separate platforms, it is crucial to have a correct synchronization between them and, in achieving this, it is necessary to take into account the implementation of the software under test. Moreover, as shown in Figure 2, in the proposed hardware-in-the-loop framework, the simulator is one of the components of the system and the interaction among them is to be done considering timing issues. Indeed the realism of the simulation depends not only on the correctness of the computed values, but also on the time at which they are provided.

[3]http://qgroundcontrol.com/ (QGroundControl - Drone Control)
[4]https://www.unrealengine.com/ (Unreal Engine 4)

As far as the synchronization problem is considered, a possible approach is to let the autopilot board trigger the simulator (*board driven* synchronization), as done by Pollini et al. [9]. Another approach, adopted by the *PX4 Flight stack* [16], is to make the simulator responsible for triggering the exchange of messages (*simulator driven* synchronization). This solution is based on a publisher/subscriber middleware and a chained execution pattern optimized to reduce the control latency. The timing of this chain is given by the publication of new sensor data achieved by means of high precision timers.

Depending on the kind of synchronization used, the simulator should provide different features. In case of a board driven synchronization, low-latency response is requested to model the dynamics with high fidelity, that is, without introducing extra delay. In case of a simulator driven synchronization, it is still important to provide low latency, but it is also necessary to ensure periodicity and regularity of the simulation cycle to reproduce the output of sensors managed by high precision timers.

In order to provide the previous features, the hardware-in-the-loop development framework has been designed with a multi-thread architecture, and an example of detailed implementation with two vehicles is shown in Figure 3. The architecture includes four different kinds of threads:

- *Inflow thread*
  This thread reads data from the autopilot board via the selected communication channel and then routes each message to the corresponding recipient (e.g., simulator, ground station).
- *Simulator thread*
  This thread simulates the dynamics of the vehicle and its sensors. It reads the control value updated by the Inflow thread and the external reactions of the environment updated by the synthetic environment; then, it computes the new vehicle state and sensors values, which will be sent directly to the autopilot board. Depending on the synchronization method used, the thread can perform a busy wait for the new data from the Inflow thread, which triggers the simulation to ensure low latency, or can freely run and sending sensor data to the board with high precision and accuracy.
- *Ground Station thread*
  This thread manages the messages exchange between the ground station and the vehicles connected to it.
- *Synthetic Environment thread*
  This thread manages the data exchange between the simulator threads and the synthetic environment application. It reads the state of the vehicles and it updates the information about the interaction with the virtual environment.

For each connected vehicle an Inflow thread and a Simulator thread are created. The design choice of separating threads for different vehicles has been undertaken to improve scalability and reduce blocking time due to shared resources.

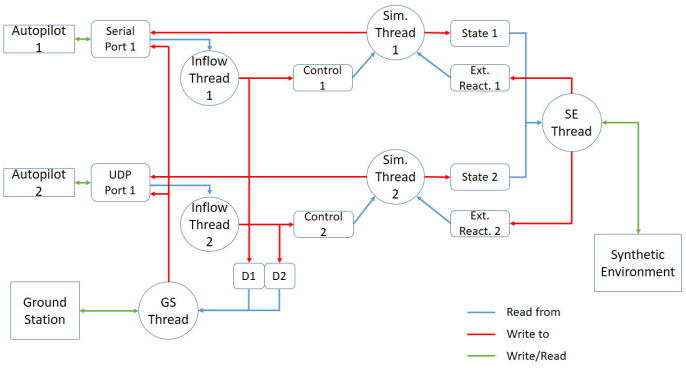

Fig. 3. Structure of the simulation software with two vehicles.

V. Experimental Evaluation

This section reports some experiments that have been carried out to characterize the system performance in terms of delays introduced by the simulation structure and achievable timing accuracy. The experiments consist in the measurement of the latency introduced by the simulator framework and its capability to produce data with a given frequency.

A. Experimental Setup

The proposed simulation framework has been employed to simulate a system with multiple quadrotor vehicles, all running the *PX4 Flight Stack* firmware. The simulations lasted 30 minutes each. The components of the simulation system were allocated on different PCs. A dedicated PC, equipped with a $Intel(R)\ Core(TM)\ 2\ Duo$ - $E8500\ @\ 3.16Ghz$ CPU and $3GB$ of RAM, hosted the core components, that is, the message routing and the simulator, whereas the ground station and the synthetic environment were running on another PC connected via Ethernet.

Three kinds of setup have been used in the experiments. In the first one, three autopilot boards (two *Raspberry 2* extended with the *Navio+*[5] modules and a *Pixhawk*[6]) were employed. The *Raspberry 2* boards hosted the *Navio+* version of the *Raspbian OS*, with a real-time patched Linux kernel, and were connected to the simulation framework via Ethernet on a dedicated LAN to reduce the jitter and the latency due to external network traffic, while the *Pixhawk* was connected via serial port.

In the other two experiments the simulation framework has been connected also to additional instances of the autopilot software, executed on a dedicated PC on the same LAN to verify the scalability property of the system. The number of the supplementary instances in these two tests was 7 and 12, respectively.

In order to acquire time measurements in a precise and non invasive way, the free I/O pins of one *Raspberry Pi 2* board were connected to a logic analyzer and toggled at specific events. Figure 4 illustrates an example of execution where τ_{sim}

[5]https://emlid.com/ (Navio Linux autopilot on Raspberry 2)
[6]http://px4.io/ (PX4 Pro Open Source Autopilot)

is the periodic simulation thread, with period T_{sim}, while τ_{ctr}, τ_{snd} are the control and the communication tasks running on the autopilot board, respectively. The measured events were the reception time t_{sns} of a new sensor data, the actuation time t_c, and the instant t_{cs} when the control data is sent to the simulator. The latency has been computed as the difference

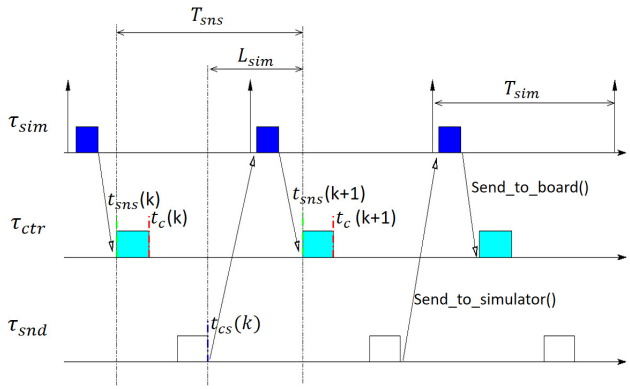

Fig. 4. Example of execution behavior.

between the time at which the actuation message is sent to the simulator and the time at which the corresponding sensor data is received back, that is $L_{sim} = t_{sns}(k+1) - t_{cs}(k)$. To measure the precision of the simulator, the inter-arrival time of sensor data ($T_{sns} = t_{sns}(k+1) - t_{sns}(k)$) has been measured. Also the actuation time ($t_c(k)$) has been recorded to check the behavior of the autopilot board while interacting with the simulator.

Two kinds of tests have been carried out under both the simulation driven synchronization and the board driven one. The former gives an insight on the periodicity capability of the proposed framework, whilst the latter allows assessing the simulation latency of the bare simulation framework.

The simulation latency, measured with 3, 10 and 15 vehicles, exhibited a linear trend with respect to the number of vehicles (see Table I). However, the differences in performance between the three cases resulted to be negligible, with a simulation latency variation in the order of 5 μs, confirming the good scalability property of the simulation framework. Due to the marginal differences in the performance and for lack of space, only the results of the experiment with 15 vehicles are reported here.

Num. Vehicles	3	10	15
Latency mean value	0.445 ms	0.448 ms	0.450 ms
Latency Std	0.089 ms	0.091 ms	0.101 ms

TABLE I
LATENCY AS A FUNCTION OF THE NUMBER OF CONNECTED VEHICLES

B. Experimental Results

The test using the simulator driven synchronization approach allowed us to verify the capability of the simulation framework to periodically trigger the autopilot board with sufficient precision. With a sending period set to 4 ms, the achieved mean value for the sensor data inter-arrival times was 4.001 ms with a standard deviation of 0.022 ms. Figure 5 presents the distribution of the inter-arrival times between successive sensor messages. The distribution of the latency

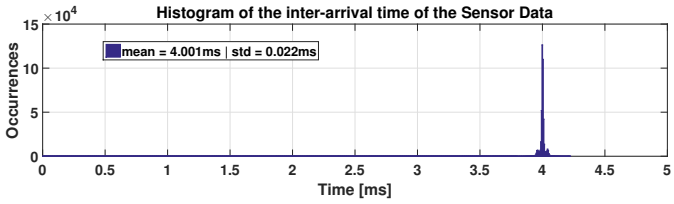

Fig. 5. Characterization of the simulation framework periodicity with the simulator driven synchronization approach.

of the simulation framework is instead shown in Figure 6, resulting in a mean value of 1.781 ms and a standard deviation of 0.723 ms. Such a high standard deviation was unexpected

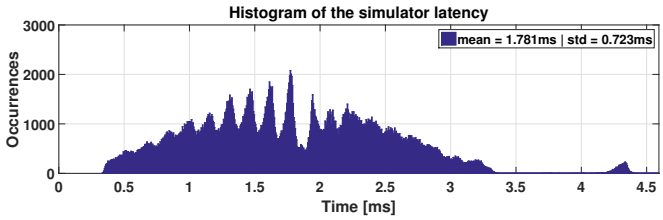

Fig. 6. Latency of the simulation framework when using the simulator driven synchronization approach.

in consideration of the design of the autopilot software [16], and thus further tests were performed to understand the source of such a variability. In particular, the control latency on the autopilot board ($L_{ctr} = t_c(k) - t_{sns}(k)$) was monitored and, as shown in Figure 7, it was seen that control messages were evaluated after several microseconds, but there was a significant and quite varying delay in sending them to the simulator. Therefore, we concluded that the observed variability is mainly due to such a transmission latency introduced by the autopilot software (which is not under our control).

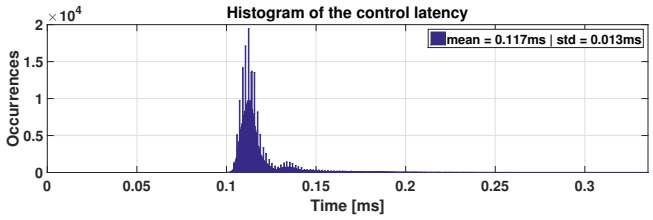

Fig. 7. Latency from publication of the new sensor topic to the control signal computation.

As a consequence, the timing behavior of the system under the simulator driven approach cannot be fully characterized from the measurements of the latency L_{sim}, since its timing behavior also depends on the implementation of the communication on the autopilot. A better test for characterizing the latency property of the proposed framework comes from

the measurement under the board driven synchronization approach. The results reported in Figure 8 show that the data is given back to the board after an interval of time with a mean value of $0.450\ ms$ and a standard deviation of $0.101\ ms$. These results were also consistent with the minimum latency observed in Figure 6.

Figure 9 reports the inter-arrival time of the sensor data when using the board driven approach. In this case, it has been noticed that the frequency at which sensory data were sent was not accurate. As in the previous case, by inspecting the control latency L_{ctr}, we concluded that the variability observed in the timing behavior was also due to the implementation of the communication on the autopilot board.

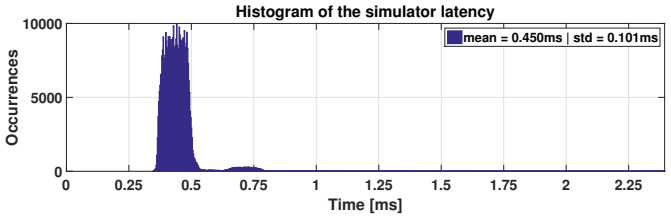

Fig. 8. Latency of the simulation framework with the board driven synchronization approach.

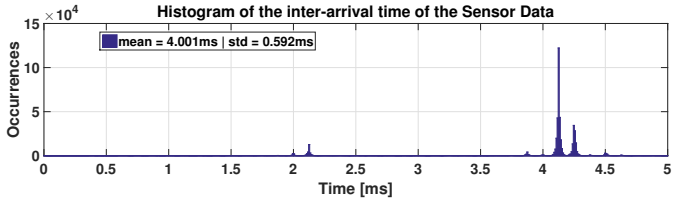

Fig. 9. Characterization of the simulation framework periodicity with the board driven synchronization approach.

VI. Conclusions

This paper presented a hardware-in-the-loop development framework aimed at simplifying and supporting the development of multi-UAV applications. Design choices have been explained and tests were performed to characterize it in terms of timing behavior.

The experimental results showed that it is possible to achieve good performance in terms of timing accuracy and precision, together with a simulation latency sufficient for common control problems. The maximum simulation latency observed under the simulator synchronization approach was in the order of the data sending period, that is $4\ ms$, while under the board synchronization approach it was less than $0.8\ ms$, below the control task period. The timing accuracy for the simulation periodicity resulted to be in the order of $20\ \mu s$. In conclusion, the proposed framework resulted to be quite effective for analyzing the timing behavior of the firmware directly on autopilot boards, hence highlighting possible timing problems when running hardware-in-the-loop simulations.

Future extensions of the proposed framework include the capability of simulating a realistic communication between vehicles (e.g., packet loss, delays) and a better integration with the synthetic environment to enable onboard camera simulation for computer vision applications.

References

[1] R. Williams, B. Konev, and F. Coenen, "Multi-agent environment exploration with ar.drones," in *Proceedings of the 15th Annual Conference on Advances in Autonomous Robotics Systems TAROS 2014*, September 1–3 2014, pp. 60–71.

[2] P. Scerri, T. Von Gonten, G. Fudge, S. Owens, and K. Sycara, "Transitioning multiagent technology to uav applications," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track AAMAS*, 2008, pp. 89–96.

[3] L. Merino, F. Caballero, J. Martinez-de Dios, I. Maza, and A. Ollero, "An unmanned aircraft system for automatic forest fire monitoring and measurement," *Journal of Intelligent & Robotic Systems*, vol. 65, pp. 533–548, January 2012.

[4] A. Posch and S. Sukkarieh, "Uav based search for a radio tagged animal using particle filters," in *Australasian Conference on Robotics and Automation (ACRA)*, Sydney, Australia, December 2009.

[5] M. Bryson, A. Reid, F. Ramos, and S. Sukkarieh, "Airborne vision-based mapping and classification of large farmland environments," *Journal of Field Robotics*, vol. 27, no. 5, pp. 632–655, 2010.

[6] Dronecode project. [Online]. Available: https://www.dronecode.org/

[7] O. Parodi, L. Lapierre, and B. Jouvencel, "Hardware-in-the-loop simulators for multi-vehicles scenarios: survey on existing solutions and proposal of a new architecture," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2009, pp. 225–230.

[8] C. Kamali and S. Jain, "Hardware in the loop simulation for a mini uav," in *4th IFAC Conference on Advances in Control and Optimization of Dynamical Systems ACODS 2016*, vol. 49, Tiruchirappalli, India, Feb. 2016, pp. 700–705.

[9] L. Pollini, V. Parnenzini, and M. Innocenti, "Distributed real-time hardware- and man-in-the-loop simulation for the icaro ii unmanned systems autopilot," in *Latest Trends in Information Technology/Recent Advances in Computer Engineering Series 7*, 2012, pp. 420–427.

[10] S. R. Barros dos Santos, S. Givigi, C. L. J. Nascimento, and N. Oliveira, "Modeling of a hardware-in-the-loop simulator for uav autopilot controllers," in *Proceedings of the 21th Brazilian Congress of Mechanical Engineering (COBEM 2011)*, Natal, RN, Brazil, October 24–28, 2011.

[11] I. Lugo-Cardenas, S. Salazar, and R. Lozano, "The mav3dsim hardware in the loop simulation platform for research and validation of uav controllers," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2016, pp. 1335–1341.

[12] Mavlink protocol. [Online]. Available: http://qgroundcontrol.org/mavlink/start

[13] M. Seleckỳ and T. Meiser, "Integration of autonomous uavs into multi-agent simulation," *Acta Polytechnica*, vol. 52, no. 5, 2012.

[14] K. S. Ali and J. L. Shumaker, "Hardware in the loop simulator for multi agent unmanned aerial vehicles environment," *American Journal of Engineering and Applied Sciences*, vol. 6, pp. 172–177, 2013.

[15] E. R. Mueller, "Hardware-in-the-loop simulation design for evaluation of unmanned aerial vehicle control systems," in *Proceedings of the AIAA Modeling and Simulation Technologies Conference and Exhibit*, August 20–23, 2007.

[16] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 6235–6240.