Simulation and Test of UAV Tasks With Resource-Constrained Hardware in the Loop

Andrea Augello*, Salvatore Gaglio*†, Giuseppe Lo Re*, and Daniele Peri*
*{andrea.augello01, salvatore.gaglio, giuseppe.lore, daniele.peri}@unipa.it
* Department of Engineering, University of Palermo, Viale delle Scienze, Ed. 6, 90128 Palermo, Italy
†ICAR-CNR, 90146 Palermo, Italy

***Abstract*—Simulations are indispensable to reduce costs and risks when developing and testing algorithms for unmanned aerial vehicles (UAV) especially for applications in high risk scenarios like search and rescue (SAR) operations and post-disaster damage assessment. Many UAV applications require real-time tasks for which the timeliness of computations is fundamental. However, standard simulation tools are not guaranteed to run in sync with real-time events, leading to unreliable assessments of the ability of the target hardware to perform specific tasks. In this work we present a simulation and test system able to run UAV tasks on resource-constrained target hardware possibly adopted in these applications. The system allows for hardware-in-the-loop simulations in which a virtual UAV provided with virtual sensors is controlled by the software under test (SUT) running on the target hardware, while simulated and real time are kept in sync. We provide experimental results from the execution of several increasingly difficult tasks in the system.**

I. Introduction

In an emergency scenario, it can be hazardous for human rescuers to enter a building as the layout may be unknown, and there could be to harmful substances present. Instead, unmanned aerial vehicles (UAVs) can explore and map unknown emergency areas also detecting potential dangers in search and rescue (SAR) operations and post-disaster damage assessment [1]. These maps can also be annotated with data from a wide array of sensors [2], both on-board and distributed in the environment, as in the case of smart buildings [3], that could also hold important information about the health status of the people inside [4].

UAV-based systems for indoor SAR missions often need to access GNSS-denied areas lacking communication infrastructure. Communication with external computers is not guaranteed, so UAVs need to operate autonomously, with enough onboard processing power for control and decision-making [5].

Developing and testing these systems in realistic physical environments can be expensive and dangerous. For this reason, simulations are indispensable tools to aid in the design and test of UAV indoor navigation and exploration systems [6].

Energy management is a crucial aspect in UAV applications [7] that imposes constraints on the choice of the onboard computer.

As not every hardware can provide the computational power needed to perform the predefined tasks correctly, hardware-in-the-loop (HIL) simulations are necessary to assess the strict constraints of these mission-critical applications accurately.

Many robot simulation systems have been used for UAVs. Some of the most popular are CoppeliaSim [8], Webots [9], and Gazebo [10]. Matlab and Simulink are also often used to simulate the kinematics of UAVs in the development of control systems due to their real-time capabilities [11], [12].

The possibility of simulating a wide array of sensors and its ease of use alongside ROS [13] make Gazebo one of the most popular simulators. Gazebo is often used in conjunction with ROS-compatible boards for HIL simulations of UAVs to evaluate the performance of specialized hardware on computationally complex tasks [14]. In addition, through the use of multiple boards with distinct ROS nodes, HIL simulations of multi-UAV systems can be performed [15]. These swarm applications for cooperative missions are often coupled with an external ground control station [16] like QGroundControl for mission coordination and planning [17]. We adopted Gazebo in this work.

An accurate simulation of message exchanges among UAVs requires a more complex approach: thoroughly testing distributed protocols in WSNs is fundamental [18]. The network characteristics in this context are very variable, so it is not straightforward to simultaneously use a network simulator and a UAV dynamics simulator. The naive solution is to run the flight simulation first, logging the position of each UAV and using those positions in a network simulator. This approach, however, is not sufficient when the received messages influence the UAV trajectory as those messages could contain high-level executable code [19], [20] to make the UAVs perform distributed processing [21]. In [22] this issue is addressed using intermediate controllers in the synchronized co-simulation of UAV flight simulators and a network simulator.

Despite its popularity, a ROS-based approach is not always optimal. Especially in multi-robot wireless contexts, such as swarms of UAVs, variability in message delay, limited bandwidth, and jitter introduce significant issues [23]. In addition, message scheduling can be an impediment to real-time operations [24]. Moreover, ROS requires the Linux operating system and is not suitable for applications with strict real-time requirements unless specific ad hoc implementations are made [25]. For these reasons, in this work, ROS is not used.

Standard simulation tools cannot accurately assess the strict time constraints typical of mission-critical applications as they do not guarantee a constant execution speed. If a HIL simulation system uses such tools, evaluating the ability of the

target hardware to meet time constraints becomes problematic. Specifically, a simulator where the simulation is slower than real-time violates the local causality constraint [26] according to which events must be processed according to their timestamp. In a simulation lagging behind real-time, commands sent from the software under test (SUT) to the UAV affect the simulation at a time before they were sent, overestimating the speed at which the target hardware completes its computations.

In this work, we present a HIL simulation and testing system that, through sensor virtualization, enables the evaluation of the ability of resource-constrained hardware to effectively complete its tasks, both in terms of correctness and timeliness. This system treats HIL simulations as distributed parallel simulations and performs synchronization operations considering the SUT and the simulator as separate logical processes (LP).

The rest of the paper is structured as follows: Section II describes the proposed system, in Section III we present the SUTs analyzed with our test system, in Section IV we try and assess how closely the simulation system matches reality, Section V shows the experimental results, finally, Section VI concludes the paper by discussing improvement areas and future development plans.

II. Proposed system

Since we are interested in evaluating whether the computational power of a target is adequate for the tasks at hand, we opt for a distributed simulation system as running the simulation on the target hardware is oftentimes not feasible and would excessively burden it. The distributed simulation system consists of a host machine, and the target hardware. The host machine runs an instance of the Gazebo [10] simulator (Fig. 1), with a virtual UAV equipped with virtual sensors and a replica of the environment the UAV is supposed to operate in.

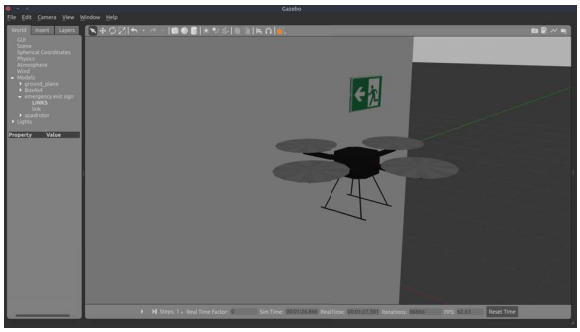

Fig. 1. A snapshot of a running simulation including a UAV and a landmark in an indoor environment.

The host machine also runs a *coordinator* program that acquires data from the simulation, provides it to the SUT, and forwards commands from the SUT to the virtual UAV. Both the coordinator and the SUT are written in C++. The SUT is executed by the target hardware, and it communicates with the coordinator via a wired TCP connection as shown in Fig. 2. Moreover, the TCP connection uses the TCP_NODELAY option, disabling Nagle algorithm to reduce jitter and delays [27]. In our setup the target hardware is a Raspberry Pi 4.

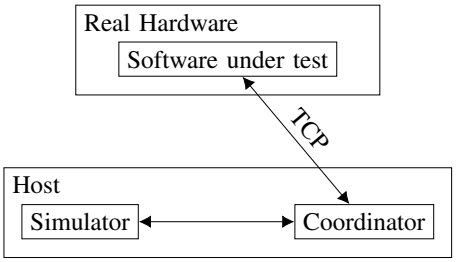

Fig. 2. Structure of the proposed system

The simulation is not guaranteed to be in sync with real time. In fact, the execution speed may not even be constant. Hence, for a realistic test of an algorithm performance on a specific target hardware a synchronization technique is needed.

In this work we opted for a conservative synchronization technique [28]. This approach avoids causality violations by preventing logical processes from processing events where it is not possible to ensure that a message with an earlier timestamp may be received in the future. In particular, barrier synchronization techniques block the execution of every LP on a specified point of execution until all of them have reached it.

In our model, the simulation time (ST) provided by the simulator acts as a global clock, providing the timestamp for every message from the coordinator. Real time is tracked separately. To associate a timestamp with every message from the SUT a timer is started when the SUT connects to the coordinator, pausing it every time the SUT is forced in an idle state to perform synchronization operations. What is being tracked, then, is the time the SUT was allowed to execute freely (ET), which would correspond to the real time if the target hardware was connected to a real UAV.

Assuming that at a certain moment ET and ST are both equal to t_i, true at least for $t_i = 0$, the simulation and the SUT can proceed independently until an interaction between the two happens when ET equals to t_{i+1}. Since the simulation will always lag behind real time, ST will be $t_{i+1} - \epsilon$ and t_{i+1} is a lower bound on the timestamp for all other future interactions.

Since ET is ahead by ϵ, every simulation step is guaranteed to be a safe event. Moreover, since all communication is initiated by the SUT, all its computation up to an interaction does not depend from the state of the simulation from t_i to t_{i+1} and can be considered safe.

As we need to enforce the local causality constraint, commands given at time t_{i+1} cannot be processed until every other event with a smaller timestamp has been processed. Doing otherwise would affect the simulation at a time preceding the moment the message was sent, and a reply cannot be sent with a timestamp preceding the request.

The execution of the SUT, alongside the ET timer, is paused until ST catches up with ET and every step of the simulation up to that point can be considered a safe event.

Since the SUT is expecting a reply from a TCP socket, this

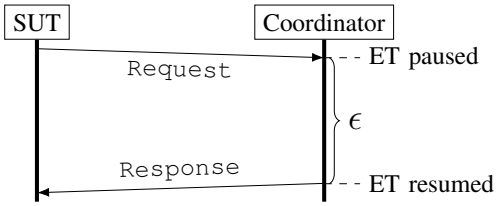

Fig. 3. General format of an interaction between the SUT and the coordinator. ϵ is measured relative to ST, not real time.

blocking call can be exploited by the coordinator to pause the execution by delaying the reply until ST reaches t_{i+1} as shown in Fig. 3.

A qualitative graph showing the trend of ST and ET is shown in Figure 4.

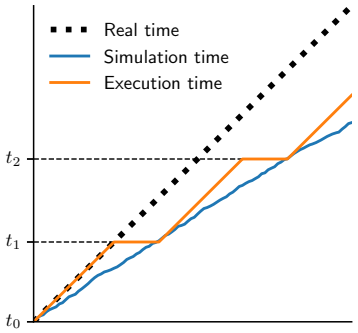

Fig. 4. Trend of ET and ST compared with real time. The marked points correspond to the interactions between SUT and coordinator.

III. Tasks

We characterize the SUT in terms of *tasks*. A task is a self-contained operation that can be split into three phases:

- an initialization phase;
- a *task loop* that repeats a sequence of operations ranging from the acquisition of sensor data to some computation on the acquired data and sending control commands to the UAV, iterated until a termination condition is reached;
- a finalization phase where, upon reaching the termination condition, control is given back to the process that started the task.

A task can also include lower-level tasks, and may start other tasks as a step of its execution in any phase.

The UAV operates in an environment populated by other entities, some of these can be known in advance and have some useful properties associated with them (e.g. shape, size, position), and can be used or interacted with during a task, as in Algorithm 2.

The execution of a task does not necessarily imply that the UAV moves or acts on the environment: it is also possible to define diagnostic tasks like the sensor characterization task detailed in Algorithm 1.

Algorithm 1 Sensor characterization task

Initialization :
Turn sensors on / connect to simulator
start_time ← get_time()
Task loop :
for $i = 1$ **to** 100 **do**
    acquire_sensor_data()
end for
Finalization :
end_time ← get_time()
return end_time - start_time

Algorithm 2 Compute relative position task

Input: anchor
Initialization :
set_tracked_object(anchor)
Task loop :
while the tracking algorithm does not converge **do**
    image ← acquire_image()
    roi ← get_object_roi(image)
end while
Finalization :
return (angle(roi), distance(roi))

The following tasks concern vision-based navigation in indoor environments. For this reason, in Algorithm 2, we define a task to compute the relative position of an object used as an anchor point through its ROI obtained using the camshift algorithm [29]. This task is a building block of other tasks. If the SUT is operating in the virtual environment the images are acquired from the simulated camera sensor (Fig. 5), when the same SUT controls a physical UAV, the camera feed is provided by an onboard sensor.

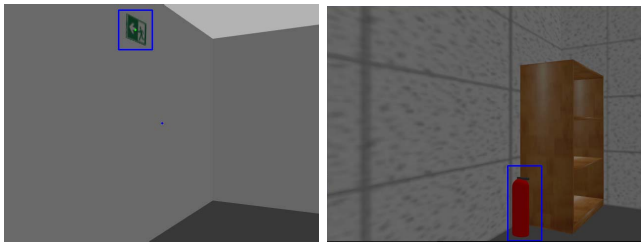

Fig. 5. Images acquired from the virtual camera, the anchor box is drawn in blue.

The "face anchor" task, described in Algorithm 3, consists in having the UAV turn until an anchor point is in the middle of the UAV camera FOV, as shown in Fig. 6, eventually reverting its rotation if it overshoots the anchor. This task is computationally intensive and the rate at which the SUT can process image samples directly reflects into the performance.

The task described in Algorithm 4 makes use of the previous task inside its loop. Here the UAV moves in a predetermined direction while keeping a line of sight with an anchor point until perpendicular to the motion direction, as shown in Fig. 7.

Algorithm 3 Face anchor task

Input: anchor, precision

Initialization :
stop()
angle ← **relative_position**(anchor).angle
rotate towards anchor
Task loop :
while angle > precision **do**
  new_angle← **relative_position**(anchor).angle
  if different_sign(angle, new_angle) **then**
    change rotation direction
    slow down
  end if
  angle ← new_angle
end while
Finalization :
stop()

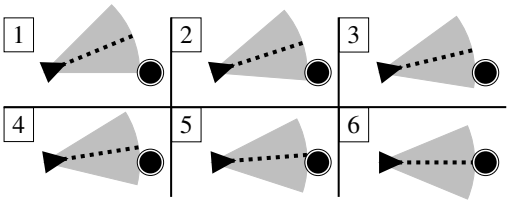

Fig. 6. "Face anchor" task. The gray area is the camera FOV. The UAV rotates towards the anchor until it occupies the center of the image.

This task can be used when trying to acquire images of an object from multiple different angles, or as a visual odometry technique.

Algorithm 4 Task to move forward while facing an anchor, until perpendicular to start direction

Input: anchor, precision

Initialization :
starting_yaw, yaw ← get_yaw()
move_direction(0)
Task loop :
while |yaw-starting_yaw| $< \pi/2$ **do**
  angle ← **relative_position**(anchor).angle
  if angle > required precision **then**
    face_anchor(anchor, precision) {lower-level task}
  end if
  yaw ← get_yaw()
  move_direction(starting_yaw - yaw)
end while
Finalization :
stop()

The tasks in the previous three algorithms assume that the anchor point is visible from the UAV. We assume indoor use and a wide enough camera angle. If it were not the case, the initialization phase should include a subtask to reach the same elevation of the anchor point to properly track it.

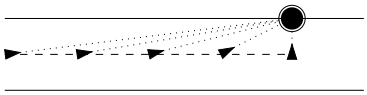

Fig. 7. "Move forward" task. The circle is an anchor point, the triangles show, from left to right, the position and orientation of the UAV in time during this task.

The task in Algorithm 5 is a diagnostic task. The UAV moves along a closed path updating its position estimate based on the IMU data. After the UAV completes the path, the SUT evaluates the error in the position estimate by measuring the relative displacement from an anchor point.

Algorithm 5 Task to move along a closed path and evaluate final positioning error

Input: anchor, distance

Initialization :
start_pos ← **relative_position**(get_image(), anchor)
move_direction(0)
Task loop :
for all angle **in**$\{0, \frac{1}{2}\pi, \pi, \frac{3}{2}\pi, 2\pi\}$ **do**
  move_direction(angle)
  $l \leftarrow 0$
  while $l <$ distance **do**
    $l \leftarrow$ update_travelled_distance(l, get_imu_data())
  end while
end for
Finalization :
stop()
end_pos ← **relative_position**(get_image(), anchor)
return distance(start_pos, end_pos)

IV. Sensor characterization

Since sensor data is sent through a TCP connection rather than being directly acquired from onboard sensor, the average time required to transfer data and the time that would be needed to acquire data from the real sensors can be very different.

As previously stated, tasks can also have a diagnostic purpose, so we used the sensor characterization task in Algorithm 1 to assess this difference.

Table I reports the comparison of the average delay when acquiring sensor data from onboard sensors with the delay when accessing the simulated ones, both on the same computer hosting the simulation and on the target hardware. The average was computed over ten experiments for each setup.

TABLE I
Sensor characterization results

Data origin	SUT location	Rangefinder	Camera
Real sensor	Target	0.0977	0.0374
Virtual	Target	0.0492	0.2510
Virtual	Host	0.0441	0.0561

The simulated rangefinder data was obtained faster than the physical equivalent, while the camera was an order of magnitude slower when accessed from the target.

By controlling the simulation execution, it can be made to proceed only for the time the data acquisition from a real sensor would have delayed the computation (Fig. 8).

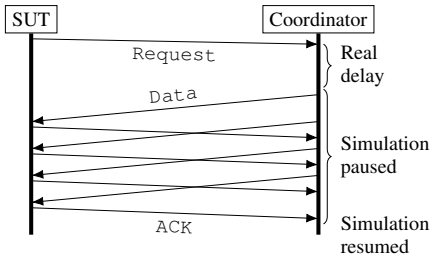

Fig. 8. Scheme to simulate the correct sensor delay.

Table II reports the performance of the "relative position" task in Algorithm 2 when executed with onboard sensors and with virtual sensors. The distance ground truth with the onboard camera is obtained with a VL53L1X infrared depth sensor [30], while in the simulated environment it is obtained directly from the simulator ground truth. In both cases the angle error was computed comparing the automatically detected ROI with a hand annotated one. The average time required was computed over the course of ten experiments for each setup.

TABLE II
PERFORMANCE OF THE "RELATIVE POSITION" TASK.

Data origin	SUT location	Error		Time
		Angle (deg)	Distance(m)	required (s)
Real sensor	Target, real-time	0.285	0.083	0.558
Virtual	Target	0.158	0.112	0.644
Virtual	Host	0.269	0.071	0.513

V. EXPERIMENTAL EVALUATION

To assess the impact of the reduced computational power on the ability of the SUT to effectively complete a given task, we performed multiple tasks with the same SUT, executing it on both the host and the target. Moreover, to assess the impact of a simulator slower than real-time on the perceived performances, we executed the SUT on the target both with and without synchronization. Every configuration was tested for ten experiments, in every experiment the ground truth from which we show the results is acquired from the simulator, moreover, the reported time is ST, not the real time.

Table III reports the performance of the SUT for the task detailed in Algorithm 3, with the anchor point located at the edge of the camera FOV. The performance was evaluated based on the time required to complete the task and on the number of times the UAV rotated too much because it could not timely detect that it had gone past the anchor. It can be seen that the reduced rate at which the SUT could track the anchor point, greatly impacted the reliability, in terms of both time required for completion and accuracy.

TABLE III
PERFORMANCE OF THE "FACE ANCHOR" TASK

SUT location	Oscillations		Time required (s)	
	mean	std. dev.	mean	std. dev.
Host	0	0	2.480	0.033
Target	1	1.612	9.242	13.767
Target, no sync	0	0	2.504	0.108

Table IV reports the performance for the task described in Algorithm 4. We evaluated the performance based on the required time to complete it and the distance from the position the simulated UAV stopped at from the correct one. The anchor is placed on a wall in a corridor at 8 meters from the start position.

In this case the time required for the task is similar in all the configurations, as it mainly depends on speed. On the other hand, the position error was noticeably smaller when the SUT was executed by the host and increased when the image-processing-heavy task was executed on the resource-constrained target. Moreover, without the forced synchronization the SUT would have shown significantly better, if misleading, performance values.

TABLE IV
PERFORMANCE OF THE "MOVE FORWARD" TASK

SUT location	Position error (m)		Time required (s)	
	mean	std. dev.	mean	std. dev.
Host	0.027	0.0187	59.689	0.800
Target	0.106	0.0440	58.484	0.517
Target, no sync	0.046	0.0454	59.014	0.960

The performance for the task described in Algorithm 5 is reported in Table V. We evaluated the performance based on the time required to complete the task, the distance from the start position at the end of the task, and the error on the position estimate.

The time required for the task depends mainly on speed, and thus little variation was measured across the setups. The estimate error was computed in static conditions, as such it is also independent from the setup. The position error shows a different pattern from all the previous measurements: the bottleneck was the 20 Hz update rate of the IMU. The task loop is very lightweight, hence the reduced computational power of the target did not have a negative impact on the performance.

VI. CONCLUSIONS

In this work we showed a limitation of most current HIL systems relying on an external simulator and proposed a method to overcome this issue in the case of single-threaded SUTs. Moreover, we assessed the impact that synchronization with the simulation time, or lack thereof, can have when

TABLE V
PERFORMANCE OF THE "CLOSED PATH" TASK

SUT location	Position error (m)		Estimate error (m)		Time required (s)	
	mean	std. dev.	mean	std. dev.	mean	std.dev
Host	0.320	0.256	0.218	0.131	27.816	1.242
Target	0.359	0.301	0.211	0.195	27.121	1.815
Target, no sync	0.328	0.262	0.245	0.122	26.899	1.596

evaluating the capability of some target hardware to execute a given task.

Future work will extend the system to test multiple UAVs virtualized on a single hardware target. SUTs will also be provided with energy-awareness. In addition, the testing platform will be extended to support development and test of high-level task descriptions including time constraints.

REFERENCES

[1] N. Kerle, F. Nex, M. Gerke, D. Duarte, and A. Vetrivel, "UAV-based structural damage mapping: A review," *ISPRS international journal of geo-information*, vol. 9, no. 1, p. 14, 2020.

[2] R. da Rosa, M. Aurelio Wehrmeister, T. Brito, J. L. Lima, and A. I. P. N. Pereira, "Honeycomb Map: A Bioinspired Topological Map for Indoor Search and Rescue Unmanned Aerial Vehicles," *Sensors*, vol. 20, no. 33, p. 907, Jan 2020.

[3] A. De Paola, "Intelligent Systems for Smart Building Management," in *Proceedings of Intelligent Environments 2018*, 2018, pp. 10–13.

[4] A. De Paola, P. Ferraro, S. Gaglio, G. Lo Re, M. Morana, M. Ortolani, and D. Peri, "An ambient intelligence system for assisted living," in *2017 AEIT International Annual Conference*, 2017, pp. 1–6.

[5] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 46–56, Sep 2012.

[6] O. Walker, F. Vanegas, F. Gonzalez, and S. Koenig, "A Deep Reinforcement Learning Framework for UAV Navigation in Indoor Environments," in *2019 IEEE Aerospace Conference*, Mar 2019, pp. 1–14.

[7] X. Cao, J. Xu, and R. Zhang, "Mobile edge computing for cellular-connected UAV: Computation offloading and trajectory optimization," in *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2018, pp. 1–5.

[8] E. Rohmer, S. P. N. Singh, and M. Freese, "CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013, www.coppeliarobotics.com.

[9] O. Michel, "Cyberbotics Ltd. Webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[10] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.

[11] P. Lu and Q. Geng, "Real-time simulation system for UAV based on Matlab/Simulink," in *2011 IEEE 2nd international conference on computing, control and industrial engineering*, vol. 1. IEEE, 2011, pp. 399–404.

[12] R. Grepl, "Real-Time Control Prototyping in MATLAB/Simulink: Review of tools for research and education in mechatronics," in *2011 IEEE International Conference on Mechatronics*, 2011, pp. 881–886.

[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[14] E. Moréac, E. M. Abdali, F. Berry, D. Heller, and J.-P. Diguet, "Hardware-in-the-loop simulation with dynamic partial FPGA reconfiguration applied to computer vision in ROS-based UAV," in *2020 International Workshop on Rapid System Prototyping (RSP)*, Sep 2020, pp. 1–7.

[15] M. Odelga, P. Stegagno, H. H. Bülthoff, 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)*, Nov 2015, pp. 204–210.

[16] Z. Zhang, W. Yang, Z. Shi, and Y. Zhong, "Hardware-in-the-loop Simulation Platform for Unmanned Aerial Vehicle Swarm System: Architecture and Application," in *2020 39th Chinese Control Conference (CCC)*, Jul 2020, pp. 58–64.

[17] C. Ramirez-Atencia and D. Camacho, "Extending QGroundControl for Automated Mission Planning of UAVs," *Sensors*, vol. 18, no. 7, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/7/2339

[18] A. Augello, R. D'Antoni, S. Gaglio, G. Lo Re, G. Martorella, and D. Peri, "Verification of Symbolic Distributed Protocols for Networked Embedded Devices," *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, vol. 2020-September, pp. 1177–1180, 2020.

[19] S. Gaglio, G. Lo Re, G. Martorella, and D. Peri, "A fast and interactive approach to application development on Wireless Sensor and Actuator Networks," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–8.

[20] S. Gaglio, G. Lo Re, G. Martorella, and D. Peri, "DC4CD: A Platform for Distributed Computing on Constrained Devices," *ACM Transactions on Embedded Computing Systems*, vol. 17, no. 1, Dec. 2017. [Online]. Available: https://doi.org/10.1145/3105923

[21] S. Gaglio, G. Lo Re, G. Martorella, and D. Peri, "High-level programming and symbolic reasoning on IoT resource constrained devices," *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 150, pp. 58–63, 2015.

[22] S. Park, W. G. La, W. Lee, and H. Kim, "Devising a Distributed Co-Simulator for a Multi-UAV Network," *Sensors*, vol. 20, no. 2121, p. 6196, Jan 2020.

[23] D. Tardioli, R. Parasuraman, and P. Ögren, "Pound: A multi-master ROS node for reducing delay and jitter in wireless multi-robot networks," *Robotics and Autonomous Systems*, vol. 111, pp. 73–87, Jan 2019.

[24] Y. Suzuki, T. Azumi, S. Kato, and N. Nishio, "Real-Time ROS Extension on Transparent CPU/GPU Coordination Mechanism," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, 2018, pp. 184–192.

[25] H. Wei, Z. Shao, Z. Huang, R. Chen, Y. Guan, J. Tan, and Z. Shao, "RT-ROS: A real-time ROS architecture on multi-core processors," *Future Generation Computer Systems*, vol. 56, pp. 171–178, Mar 2016.

[26] Fujimoto, "Distributed simulation systems," in *Proceedings of the 2003 Winter Simulation Conference, 2003.*, vol. 1, Dec 2003, pp. 124–134 Vol.1.

[27] J. C. Mogul and G. Minshall, "Rethinking the TCP Nagle algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 1, pp. 6–20, Jan 2001.

[28] S. Jafer, Q. Liu, and G. Wainer, "Synchronization methods in parallel and distributed discrete-event simulation," *Simulation Modelling Practice and Theory*, vol. 30, pp. 54–73, Jan 2013.

[29] G. R. Bradski, "Real time face and object tracking as a component of a perceptual user interface," in *Proceedings Fourth IEEE Workshop on Applications of Computer Vision. WACV'98 (Cat. No.98EX201)*, 1998, pp. 214–219.

[30] R. M. Jans, A. S. Green, and L. J. Koerner, "Characterization of a miniaturized IR depth sensor with a programmable region-of-interest that enables hazard mapping applications," *IEEE Sensors Journal*, vol. 20, no. 10, pp. 5213–5220, 2020.