DANCERS: A Physics and Network Co-Simulator for Communicating Multi-Robot Systems

Théotime Balaguer, Olivier Simonin, Isabelle Guérin-Lassous, Isabelle Fantoni

DANCERS: A Physics and Network Co-Simulator for Communicating Multi-Robot Systems

Théotime Balaguer*†‡, Olivier Simonin*, Isabelle Guérin Lassous†, Isabelle Fantoni‡
*INSA Lyon, Inria, CITI, UMR 3720, Villeurbanne, FRANCE
Email: theotime.balaguer@insa-lyon.fr
†Université Claude Bernard Lyon 1, ENS Lyon, CNRS, LIP, UMR 5668, Lyon, FRANCE
‡Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, Nantes, FRANCE

***Abstract*—Multi-robot systems equipped with wireless communication devices can outperform single-robot platforms through cooperation. However, such wireless network face challenges such as the unreliable wireless channels, physical obstacles, uneven terrains, and potential malicious attacks. Studying the interplay between the physical and network aspects of these systems proves challenging because existing physics simulators often lack realistic communication models, while network simulators typically use over-simplified physics and mobility models. To address this gap, we introduce DANCERS, a novel co-simulation platform providing synchronization and information exchange between any robotic and any network simulator. DANCERS is evaluated in terms of computational performance, overhead and correctness, and its capabilities are demonstrated through a multi-robot use case.**

***Index Terms*—Co-simulation, Multi-robot Systems, Network Simulation, Simulators Synchronization.**

I. Introduction

Cooperative Multi-Robot Systems (MRS) offer new opportunities for civil or military missions such as environment surveillance, search-and-rescue or communication networks [1]. In many situations, such systems outperform single-robot platforms by bringing speed, robustness and scaling, thanks to the maturation of artificial intelligence, embedded computing and IoT network [2].

To form a robotic network, the robots involved in a cooperative MRS must be able to interact with each other, either with perception only, such as visual detection, or more frequently with radio-wave communication. Different information can be exchanged on the network, ranging from simple robot positions to heavy video streams and serving different purposes such as navigating in a complex environment while transmitting a video stream to an operator.

In the context of cooperative MRS, wireless network and robotic control are both part of the same complex system, and should be studied together [1]. Simulation plays a major role in the development of cooperative MRS to enable fast, reproducible, safe and inexpensive studies, and would greatly benefit from joint robotic and network simulation. Both the robotics and network communities have used simulation for decades and the simulators of each field are now numerous, well developed and refined [3] [4]. However, physics simulators tend to ignore networking issues while network simulators offer limited or unrealistic models of the physical world. This assessment naturally brings the idea of co-simulation — merging two simulators in a single program — to get the best of both worlds.

Several challenges arise from co-simulation, such as time synchronization, information sharing and support for *Software-in-the-loop* (SITL). SITL is the ability to test and validate the real robotic software in the simulation, which facilitates the transfer from simulation to real-life experimentation. In modern robotics, this implies support for commonly used autopilots and for the *Robotic Operating System* (ROS). ROS is a middleware commonly used in robotics, with two major versions that will be referred as ROS1 and ROS2 in the remainder of this article. ROS1 will reach end-of-life in May 2025, leaving all development efforts to ROS2.

Our study of existing physics and network co-simulators showed that they were difficult to use out-of-the-box. Indeed, maintenance is a challenge in the rapidly evolving world of robotics, creating compatibility issues between existing co-simulators and state-of-the-art tools. Up-to-date documentation is not always available, hindering the use of existing co-simulators. Hoping to fill a gap, we introduce DANCERS which stands for Distributed, Autonomous, Networked and CooperativE Robots Simulator, available under open-source licensing [5]. DANCERS is not tied to a single physics or network simulator and we provide the connectors for three physics simulators and one network simulator (ns-3 [6]). We evaluate this co-simulator against computational performance, overhead and correctness, and show a case study with a Flying Ad-hoc Network (FANET) to illustrate DANCERS' potential.

In this paper we will first present in Section II the existing work on joint physics and network simulation. In Section III we introduce our co-simulation architecture called DANCERS and present available connectors for simulators and the virtualization mode in Section IV. The co-simulator's properties are evaluated in Section V. Section VI presents a case study example showing the capabilities of DANCERS. Finally, we conclude in Section VII.

This work was jointly funded by the "Agence de l'Innovation de Défense" (AID) and "Institut National des Sciences Appliquées" (INSA Lyon), and realized within the context of the CONCERTO project (ANR-20-ASTR-0003).

II. Related work on (co-)simulators

MRS researchers studied the joint simulation of physics and network for about a decade. A selection of the most influential simulation tools is presented hereafter.

ARGoS [7] is a simulator specifically designed for MRS. Among other innovations, ARGoS natively simulate inter-agent communication, making it a solid option for the joint study of robotics and networking. However, its communication models are far from the networking community standards (*e.g.* radio model with fixed communication range, routing algorithms not simulated). Because network simulation brings great complexity, co-simulation has emerged as a preferred approach to incorporate state-of-the-art simulators such as ns-3 [6] (described in Subsection IV-A1).

RoboNetSim [8] is one of the oldest co-simulation tool. It integrates one of the two physics simulator ARGoS [7] or Player/Stage [9] with either ns-2 or ns-3. Specifically designed for large swarms, it can handle hundreds of robots. Now a decades old, RoboNetSim principles and implementation are still relevant, but it uses very old software and its adaptation to recent robotic software such as Gazebo [10] or ROS2 would be far from trivial. Despite these limitations, RoboNetSim was of great inspiration. Their use of sockets for inter-simulator communication and their evaluation method for a co-simulator were the basis of the current work.

CORNET [11] and CORNET 2.0 [12] interconnect Gazebo [10] and ns-3 [6]. It is recent, open-source and maintained. However, it uses an unidirectional time synchronization scheme that holds the strong assumption that the network simulator always runs faster than the physics simulator. This assumption does not hold for all scenarios, as it is shown in [8] and in the present work in Fig. 5. Furthermore, documentation is missing for CORNET, making it hard to apprehend.

ROS-NetSim [13] is a coordination framework based on ROS1 that merges any physics and network simulators. It uses a time-stepped synchronization approach and leverages the *Protobuf*[1] library for information exchange. The network simulator used in the examples, "WINTERSim", is no longer available. ROS-NetSim inspired the present work, specifically for the message format and processing.

FlyNetSim [14] focuses on SITL simulation, where the software of the robot is directly tested in simulation. It enables time and position synchronization between multiple instances of the "Ardupilot"[2] SITL software and ns-3. FlyNetSim cannot run faster than real time and lacks a full physics engine, as it only uses the dynamic models included in the Ardupilot SITL. FlyNetSim provided guidance in the development of our robot virtualization mode (see Subsection IV-B).

This list is not exhaustive, for a recent and complete list of the existing co-simulators and their characteristics, the reader can refer to the Table I. of [12].

[1]Protocol Buffer: https://protobuf.dev/

[2]https://ardupilot.org/

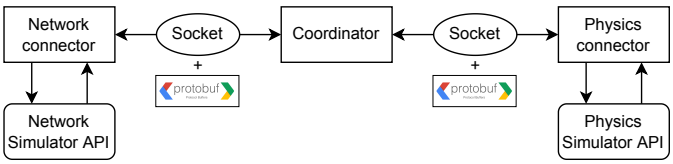

Fig. 1. High-level architecture diagram of DANCERS

III. The DANCERS architecture

DANCERS is a C++-written co-simulator designed to connect any physics simulator with any network simulator, providing synchronization and information sharing. DANCERS has three modules at its core: the *Coordinator*, the *Physics Connector* and the *Network Connector*. This simple architecture is depicted in the Fig. 1. In this section, we explore how DANCERS answers the synchronization and information exchange challenges, and we give details on how obstacles are managed.

A. Time synchronization

The co-simulation consists in a parallel execution of both simulators, with a synchronization and data exchange process occurring iteratively. Each iteration spans a fixed simulated time period, typically tens of milliseconds, configured before the simulation starts. Choosing the iteration duration is difficult because physics and network simulators operate on different time scales: physics simulators need iterations of 4–10 ms, while network simulators require iterations of 1 ms or less. On the physics simulation side, smaller time steps dramatically increase computational costs while only marginally improving accuracy, whereas larger time steps in network simulations introduce undesirable delays in data exchange.

To overcome this issue, we introduce a "decoupled time-stepped" approach, shown in Fig. 2, which divides each iteration (Δ) into smaller *steps* specific to each simulator: δ_{phy} and δ_{net}. The relationship is given by:

$$\Delta = N_{\text{phy}} * \delta_{\text{phy}} = N_{\text{net}} * \delta_{\text{net}} \tag{1}$$

where $N_{\text{phy}}, N_{\text{net}} \in \mathbb{N}$ are the number of steps per iteration for the physics and network simulators, respectively.

Using two threads, the Coordinator oversees this process, ensuring N_{phy} and N_{net} steps are executed per iteration. A step is triggered when a Connector receives a synchronization message from the Coordinator (`START`). The Connector then advances its simulation by δ_{phy} or δ_{net} and signals the end of the step by sending a synchronization message to the Coordinator (`END`). Once both Connectors have completed their required number of steps, a new iteration begins.

The operator configures Δ, δ_{phy} and δ_{net} in the configuration file, balancing accuracy and computational cost while respecting the relationship in (1).

DANCERS also manages the global ROS2 clock by updating it whenever the Connector with the smallest step size completes a step. This clock allows the ROS2 nodes in the physics simulator to synchronize with the co-simulator, with a precision of $\min(\delta_{\text{phy}}, \delta_{\text{net}})$.

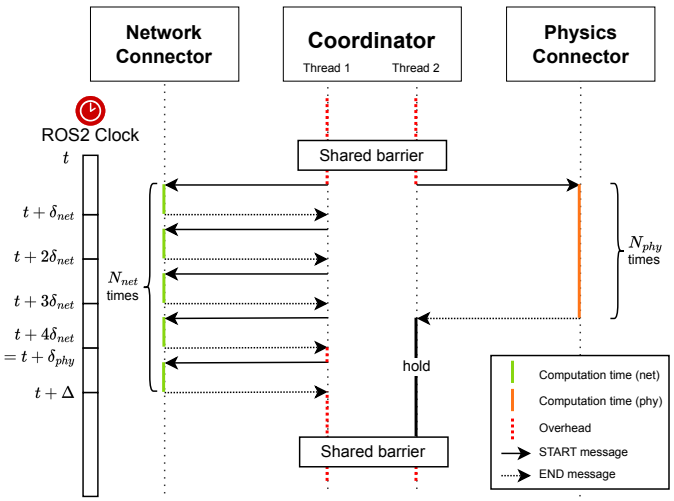

Fig. 2. The synchronization scheme for the DANCERS co-simulator

B. Information Exchange

The Coordinator and the Connectors exchange messages with sockets. The choice of sockets over other inter-process communication methods such as shared-memory or message queues have been discussed in [8] and we share the authors' conclusions: sockets are extremely flexible, with a good balance between efficiency and complexity of usage. The modules of DANCERS can be executed on a single machine (using *Unix Domain Sockets*, UDS) or distributed among interconnected machines (using *Transmission Control Protocol*, TCP). The *Protobuf* format is used to serialize and deserialize the messages, that are compressed with the `zip` algorithm before transmission. The only mandatory information that the simulators must share is robots positions. It is left to the operator to decide what other information should be shared, depending on the simulated models.

Synchronization messages can be defined as follows:

$$\begin{cases} M_{\text{phy}} = (\mathbf{p}, P_{phy}) \\ M_{\text{net}} = (P_{net}) \end{cases} \tag{2}$$

where $\mathbf{p} \in (\mathbb{R}^3)^N$ is a vector containing the robots' positions for N robots, and P_{phy}, P_{net} are operator-defined data. The nature of the synchronization messages (`START` or `END`) is implicitly defined by which module generated the message.

Exchanging messages between the different modules of the co-simulator have a computational cost called the *overhead*, which is highly correlated with the size of the synchronization messages. It is therefore important to keep the size of the operator-defined payloads as small as possible.

C. Simulating obstacles in DANCERS

In robotics, navigating in a complex environment filled with obstacles is already a challenge, but obstacles also have a significant effect on the quality of radio signals by blocking the propagation in Non-Line-of-Sight (nLOS) situations, or creating interference due to multi-path. In DANCERS, obstacles are defined in the configuration file and created at initialization both in the physics and network simulators. This limits the obstacles scene to remain static for the duration of the simulation, but reduces the size of the synchronization messages. Network simulators often offer less detailed environmental representations compared to physics simulators, which may impose further constraints on obstacles. As an example, ns-3 requires obstacles to be cuboid and oriented along the x, y, and z axes.

IV. Connectors for different simulators and virtualization mode

There is a balance between realism and complexity in any simulation. The level of realism must always be justified by the scientific question at hand, to keep complexity as low as possible [15]. It is therefore beneficial to have a broad range of simulation capabilities for different purposes. Besides, creating a Connector for a simulator requires a good understanding of its internal functioning. To cover for multiple levels of realism, we developed Connectors for three physics simulators, presented hereafter. On the network side, we only developed one Connector because ns-3 covers a broad range of protocols and technologies while being computationally efficient. DANCERS also supports SITL, which can be seen as the most advanced level of simulation, by functioning in "virtualization" mode.

A. Implemented Connectors

1) ns-3: ns-3 is a widely used network simulator supporting various communication technologies (Wi-Fi, LoRa, UWB, LTE, 5G, etc.) with high-fidelity protocol simulation across the network stack. While not specialized for detailed physical layer simulations, it offers advanced propagation models like the *3GPP Vehicle-to-Vehicle Urban Propagation Model* that accounts for distance, obstacles, and shadowing — adequate for most MRS simulations. Furthermore, ns-3 benefits from an active community that ensures diverse, well-documented, and rigorously tested modules, fostering trust among network researchers.

2) Robotsim: Robotsim [16] is an open-source physics simulator designed for studying UAV swarming. Chosen for its speed, it can simulate hundreds of robots faster than real-time, enabling the study of high-level multi-robot navigation algorithms like flocking. Its low computational cost makes Robotsim a good choice for machine learning models training or evolutionary algorithms. Additionally, it aligns with our team's previous work [17] and is illustrated in Fig. 3 (left).

3) Mini-dancers: Mini-dancers is a multi-UAV simulator built on ROS2 and the multi-rotor model developed by the Czech Technical University's MRS Group [18]. It uses Rviz2 for 3D visualization (illustrated in the middle of Fig. 3). While obstacles can be visualized, collisions are not simulated.

Mini-dancers strikes a balance between Robotsim and Gazebo: it scales to hundreds of UAVs with lower computational demands than Gazebo and offers modern features compared to Robotsim. In DANCERS, it is ideal for studying cooperative multi-UAV systems, when the scientific question

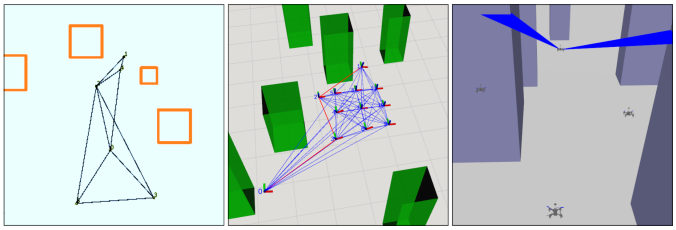

Fig. 3. Visual interface of the three physics simulators available for DANCERS, from the less complex (left) to the most complex (right). On the left, Robotsim with 6 UAVs. In the middle, Mini-dancers with 12 UAVs. On the right, Gazebo with 4 UAVs (PX4Vision model) equipped with lidars.

does not require the simulation of autopilot software, realistic sensors and collisions.

4) Gazebo: Gazebo [10] is a broadly adopted open-source physics simulator for robotic systems. It supports various robot types and environments, offering a rich library of sensors, actuators, and a large user community. An example of UAVs navigating a cluttered environment with simulated LIDARs is shown in Fig. 3 (right).

In DANCERS, using Gazebo allows for realistic simulations of both robotics and networking, closely approximating real-world scenarios. However, it is usually paired with SITL autopilots and simulated sensors, whose high computational demands limit scalability to a few dozen robots per machine.

B. Network virtualization mode

The ability to test MRS innovations in a SITL manner — using the same code (nodes and OS) as on the real robot, even for the network stack — is highly valuable but rarely implemented. It requires the OS-managed network stack of each robot to be virtualized, enabling onboard nodes to exchange information over a simulated network as if it was running on real hardware. In this DANCERS mode, network virtualization (specifically *network namespaces* (NNS), a standard UNIX feature) replaces part of the network stack previously simulated by the network simulator. The network stack is thus *virtualized* for the network – IP – layer and higher, and *simulated* for the link – MAC – and physical – PHY – layers. The network architecture and layer responsibilities are illustrated in Fig. 4.

When a robotic node sends a message, the virtualized network stack processes the packet down to the IP layer. Afterward, the network simulator computes radio transmission effects. If successful, the packet is converted back for processing in the destination robot's virtualized network stack, and finally received by the destination node. Some simulation-specific data has to be extracted from the robotic nodes (*e.g.,* the motor rotation speed) without interfering with the inter-robot communications. To separate these flows from simulated communication, a dedicated interface routes this traffic out of the NNS (*veth 2* in Fig.4).

This mode of DANCERS requires the network simulator to be able to get and return MAC layer packets from and to the user-space of the OS. For example, this can be done using ns-3's *TAP-Bridge* module.

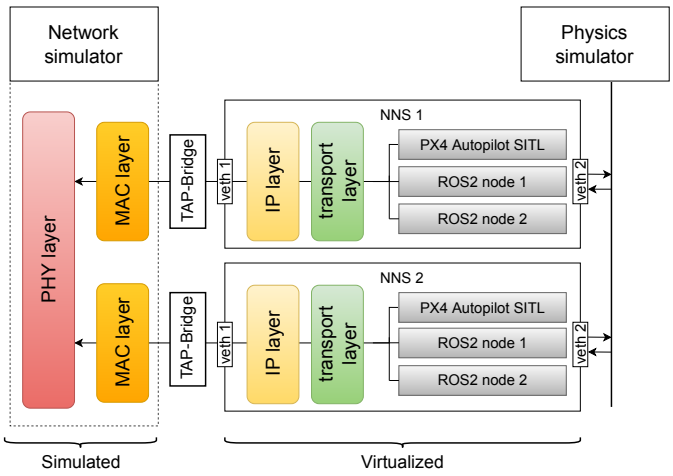

Fig. 4. Network virtualization framework for DANCERS virtualization mode

V. DANCERS CO-SIMULATOR EVALUATION

Following the method proposed in [8], DANCERS is evaluated with three key characteristics: computational performance, overhead and correctness. We chose a standard MRS scenario to run the evaluation, described in details in Sec. VI, where a fleet of collaborative UAVs is tasked to navigate autonomously across an environment with obstacles using a distributed *flocking* algorithm [17]. The simulations were run on a laptop computer with the following characteristics: Intel Core i7-12700H CPU (20 cores), NVIDIA RTX A2000 (8GB) Laptop GPU, 32GB DDR5 4800MHz RAM, and Ubuntu 22.04 LTS (64-bit) OS.

A. Computational performance

Computational performance is one of the most important features of any simulator. We evaluate it with the *simulation-time factor* (STF), the ratio between the computation time (wall clock) and the simulated time:

$$STF = \frac{T^{\Delta}}{\Delta} \tag{3}$$

where T^{Δ} is the computation time for one iteration and Δ is the length of one iteration. The STF can be calculated for any module, for example, the STF of the physics simulator would be T^{Δ}_{phy}/Δ where T^{Δ}_{phy} is the computation time of the physics simulator.

Fig. 5 shows the STF of the two simulators and the entire co-simulation against the number of simulated robots. It shows values for the Robotsim variation (left), the Mini-dancers variation (center) and the Gazebo variation (right). The values are averaged on the entire scenario phase (100 s). The first important fact to notice is that the three variations of the co-simulator run faster than real-time when the number of robots is below 20 robots approximately. We can clearly see the efficiency of Robotsim to simulate large groups of robots, keeping a low computational footprint even with 64 robots. The Robotsim variation also showcases that the network simulator can be the slowest of the two simulators, with

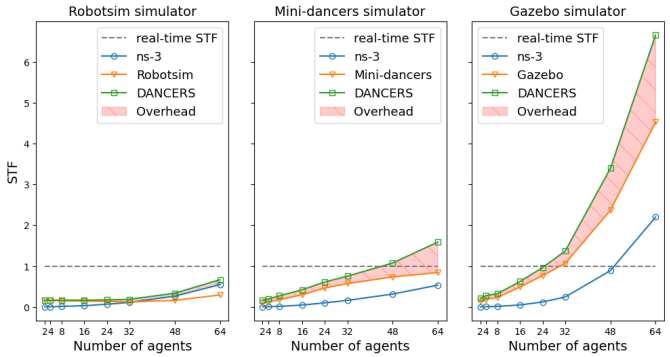

Fig. 5. STF of DANCERS' modules and for three physics simulators: Robotsim (left), Mini-dancers (center) and Gazebo (right)

the computational footprint of ns-3 overgrowing the one of Robotsim. The increase of the STF with large numbers of robots with Gazebo can be explained by the high CPU usage during the experiment, which was monitored at 97%. Indeed, in Gazebo each robot is controlled by an independent instance of the PX4 Autopilot SITL, which is demanding in CPU resources and slows the other processes.

B. Overhead

The *overhead* represents the computational cost of co-simulation. It is caused by the unavoidable synchronization messages exchange between the two simulators and usually depends on the amount of shared data. For example, in DANCERS, the overhead consists in a large part of the serialization, deserialization, compression and decompression of the Protobuf messages conveying information between the two simulators. We define the overhead as the difference of computation time between the co-simulator and the slowest of the two simulators:

$$T^{\Delta}_{\text{overhead}} = T^{\Delta} - \max(T^{\Delta}_{\text{phy}}, T^{\Delta}_{\text{net}}) \tag{4}$$

where T^{Δ}_{phy} and T^{Δ}_{net} are the computation times for one iteration by each simulator and T^{Δ} is the total computation time for one iteration. $T^{\Delta}_{\text{overhead}}$ is then the time when both simulators are inactive during one iteration.

The overhead depends on two factors: the number of exchanged synchronization messages and their length. Small step sizes (δ_{phy} and δ_{net}) increase the number of synchronization messages. The size of these messages depends on the number of robots and on the size of the operator-defined payloads (P_{phy} and P_{net}). In Fig. 2, the overhead appears in dashed red in an example where the network simulator is slower than the physics simulator. The overhead STF ($T^{\Delta}_{\text{overhead}}/\Delta$) is also pictured in Fig. 5 in hatched red.

Better than the absolute value of the overhead cost, the user might be interested in the share of the overhead in the total computation cost ($T^{\Delta}_{\text{overhead}}/T^{\Delta}$). For example with Gazebo, 32 robots and $\Delta = \delta_{phy} = \delta_{net} = 10\,\text{ms}$, the overhead represents on average 21% of the computation time for one iteration. When the number of robots increases or the iteration length decreases, the absolute value of the overhead grow but its ratio relative to the total computation cost stays around 20%. We deem this co-simulation cost acceptable, yet reducing it would be a good trail for future work.

C. Correctness

A co-simulator is deemed *correct* if it does not introduce unexpected errors, meaning that a co-simulation produces the same results than standalone simulators, provided with the same inputs. To assess the correctness of DANCERS, we simulate a scenario where the network has no effect on the robotic control and compare the results given by DANCERS with the results of the independent simulators. The Mini-Dancers physics simulator paired with ns-3 was used for this evaluation.

In both cases, the UAVs' positions are *exactly* the same: at each iteration, the x, y and z values match perfectly. Similarly, packet arrival times are identical across co-simulation and ns-3 only. This consistency was achievable because both the UAV model in Mini-Dancers and ns-3 provide simulation reproductibility through pseudo-random number generators seeded with the save value in both experimental setups. The data generated for the correctness evaluation can be consulted on DANCERS' online repository [5].

Note that it would be tremendously difficult to prove the correctness of DANCERS' robot virtualization mode, because it does not provide simulation reproducibility due to its interactions with the "real" world.

VI. Flocking Case Study

This case study shows that DANCERS can simulate the dependency between the robotic behaviors and their communication. Flocking is a well known method for multi-robot navigation where each robot locally computes its command based on the position and velocity of its neighbors. To select these neighbors, most of the flocking literature rely on an interaction range either arbitrarily defined [19] [20] or defined by a more or less sophisticated communication models that only partially represent the complex networking interactions between robots [21] [22]. For example, effects of obstacle shadowing on flocking have been studied in [17] and reducing the communication costs of flocking has been studied in [23], but never with a comprehensive network simulator, to the best of our knowledge. DANCERS allows to study the flocking problem with realistic network protocols.

Scenario: A fleet of 4-UAVs autonomously navigates through an obstacle-filled environment. Each UAV, equipped with a simulated lidar, executes the Vásárhelyi with Attraction (VAT) flocking algorithm [17] to determine its trajectory. UAVs are simulated in Gazebo using the PX4 autopilot, while communications are handled in ns-3.

Each UAV receives perfect localization data (position and velocity) from the physics simulator and broadcasts this state via an ad-hoc Wi-Fi network using the UDP protocol at a configurable State Broadcast Rate (SBR). Received neighbor states are stored with a lifetime of τ seconds; neighbors are disregarded if their latest state exceeds this time threshold.

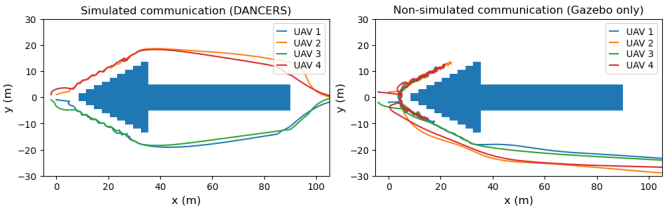

Fig. 6. Trajectories of 4 UAVs crossing an arrow-shaped obstacle, moving from left to right. On the left, communications are simulated with ns-3 (DANCERS). On the right, communications ignore obstacles, *i.e.,* not realistic.

UAVs are attracted to a global target placed infinitely far on the opposite side of the environment.

Results: We show that signal attenuation caused by obstacles drastically impacts the behavior of a communication-based flocking. We compare the results of DANCERS against standard Gazebo without network simulation, where communication is near-instantaneous, and message loss is minimal. Fig. 6 shows the top-view trajectories of four robots encountering an arrow-shaped obstacle designed to clearly show the effect of obstacle shadowing on communication-based flocking. When the fleet encounters the obstacle, it splits into two groups with obstructed line-of-sight. With properly simulated communication (*i.e.* with ns-3), the obstacle blocks most of the messages, causing the subdivision of the fleet in two subgroups after $\tau = 2$ s. Each subgroup operates independently until the end of the obstacle is reached, then connectivity is regained and the fleet reunites. In contrast, with plain Gazebo, communication remains unaffected by the obstacle. The UAVs maintain mutual attraction, leading one subgroup to turn back, rejoin the other and get around the obstacle without fragmentation. A video of a similar scenario is available online[3].

In real life, an obstacle of this size would likely block radio signals, leading to a temporary disconnection and the fleet behavior simulated by DANCERS. Co-simulation enables such network-realistic studies, that could not be produced without a joint robotic and network simulation. Additional studies using other physics simulators (Mini-Dancers, Robotsim) are omitted here for brevity.

VII. Conclusion

We introduce DANCERS, a co-simulator able to interconnect any physics simulator with any network simulator, with the goal of accelerating the study of communicating multi-robot systems (MRS). Different modes covering different simulation purposes in terms of realism and complexity of computational load are provided. We show that the co-simulator's performances are comparable with the ones of the underlying simulators, the overhead imposed by synchronization is acceptable and the simulation correctness is preserved. Through a multi-UAV use case, we illustrate how DANCERS can be used to accurately simulate cooperative MRS navigating among obstacles.

[3]https://youtu.be/FubicGPyB1E

DANCERS is open-source under the GPLv3 copy-left licensing. Its source code and the necessary code to reproduce the simulations presented in this work are available at https://github.com/Chroma-CITI/DANCERS. We also provide containerized installation (with Docker) and documentation. In the future, we aim to extend the evaluation of DANCERS and explore more elaborate multi-robot scenarios.

References

[1] S. Hayat, E. Yanmaz, and R. Muzaffar, "Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint," *IEEE Communications Surveys & Tutorials*, 2016.

[2] M. Mozaffari *et al.*, "A tutorial on UAVs for wireless networks: Applications, challenges, and open problems," *IEEE Communications Surveys & Tutorials*, 2019.

[3] J. Collins *et al.*, "A Review of Physics Simulators for Robotic Applications," *IEEE Access*, 2021.

[4] M. H. Kabir *et al.*, "Detail Comparison of Network Simulators," *International Journal of Scientific & Engineering Research*, no. 10, 2014.

[5] T. Balaguer, "DANCERS," 2024. [Online]. Available: https://github.com/Chroma-CITI/DANCERS

[6] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010.

[7] C. Pinciroli *et al.*, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, 2012.

[8] M. Kudelski, L. M. Gambardella, and G. A. Di Caro, "RoboNetSim: An integrated framework for multi-robot and network simulation," *Robotics and Autonomous Systems*, 2013.

[9] B. Gerkey *et al.*, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *ICAR*, 2003.

[10] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IROS*. IEEE, 2004.

[11] S. Acharya *et al.*, "Cornet: A co-simulation middleware for robot networks," in *COMSNETS*, 2020.

[12] S. Acharya *et al.*, "A co-simulation framework for communication and control in autonomous multi-robot systems," in *IROS*. IEEE, 2023.

[13] M. Calvo-Fullana *et al.*, "ROS-NetSim: A framework for the integration of robotic and network simulators," *IEEE Robotics and Automation Letters*, 2021.

[14] S. Baidya, Z. Shaikh, and M. Levorato, "FlyNetSim: An Open Source Synchronized UAV Network Simulator based on ns-3 and Ardupilot," in *MSWIM*. ACM, Oct. 2018.

[15] B. Zeigler, H. Praehofer, and T. Kim, *Theory of Modeling and Simulation*. Elsevier Science, 2000.

[16] C. Virágh, "Robotsim multi-robot simulator," 2018. [Online]. Available: https://github.com/csviragh/robotsim

[17] A. Bonnefond, O. Simonin, and I. Guérin-Lassous, "Extension of flocking models to environments with obstacles and degraded communications," in *IROS*. IEEE, 2021.

[18] T. Baca *et al.*, "The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles," *Journal of Intelligent & Robotic Systems*, Apr. 2021.

[19] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: algorithms and theory," *IEEE Transactions on Automatic Control*, 2006.

[20] G. Vásárhelyi *et al.*, "Optimized flocking of autonomous drones in confined environments," *Science Robotics*, 2018.

[21] H. Li *et al.*, "Flocking of mobile agents using a new interaction model: A cyber-physical perspective," *IEEE Access*, 2017.

[22] T. Ibuki *et al.*, "Optimization-based distributed flocking control for multiple rigid bodies," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, 2020.

[23] M. M. Zavlanos and G. J. Pappas, "Distributed connectivity control of mobile networks," *IEEE Transactions on Robotics*, vol. 24, no. 6, 2008.