FT2025
SWEDISH AEROSPACE AT THE CROSSROADS
The 12th Swedish Aerospace Technology Congress
FT2025 in Stockholm
October 14-15, 2025

A New Multi-Agent Simulator Framework Using Hopsan and Unreal 5.3

Andrew Gomes Pereira Sarmento[1], Robert Braun[2], Petter Krus[2], and Emilia Villani[1]

[1]*Department of Aeronautical and Mechanical Engineering, Aeronautics Institute of Technology (ITA), São José dos Campos, Brazil*
E-mail:, andrewgps@ccm-ita.org.br , evillani@ita.br
[2]*Department of Management and Engineering, Linköping University (LiU), Linköping, Sweden*
E-mail:, robert.braun@liu.se , petter.krus@liu.se

Abstract

Unmanned Aircraft Systems (UAS) research increasingly requires high-performance, multi-agent simulation environments that integrate realistic dynamics with immersive visualization. This paper introduces a distributed co-simulation framework that seamlessly combines the dynamic modeling capabilities of Hopsan with the advanced rendering and interaction tools of Unreal Engine 5.3. Communication between the two platforms is achieved through a lightweight, User Datagram Protocol (UDP) based plugin, which supports bi-directional real-time data exchange and is complemented by a USB Raw Input plugin to integrate human-in-the-loop joystick control. The proposed framework was validated across progressively complex scenarios. First, a single F-16 aircraft data model was imported from Hopsan, encompassing waypoint-guided dynamics, atmospheric effects, actuation, and geoid-based altitude calibration. Its state reinforced by Hopsan was visualized in Unreal in real-time. Second, the platform was extended to support two independent F-16 agents, each communicating via dedicated UDP ports, thereby demonstrating modular, scalable, multi-agent operation. Third, we introduced a Human Machine Interface (HMI) scenario, where one aircraft was piloted manually via joystick input, while the second autonomously followed the same waypoint sequence. This validated the framework's capacity to handle human interaction in a multi-agent context. Results evidence synchronized simulation at real-time performance, accurate environmental interactions (terrain, wind, collision), and reliable human-in-the-loop control. The framework's architecture promotes modularity, scalability, and deployment flexibility across multiple machines. Future enhancements will explore tighter coupling with Unreal's environmental physics, adoption of fluid dynamics, and scaling to larger agent ensembles. By integrating open-source dynamical modeling with high-fidelity graphical simulation, this platform offers a robust foundation for UAS mission planning, operator training, and AI-driven control validation.

Keywords: Multi-Agent Simulation, Co-Simulation Architecture, Unmanned Aircraft Systems (UAS), High-Fidelity Visualization

1 Introduction

Unmanned Aerial Vehicles (UAVs) have become indispensable tools in both military and civilian domains. In the civil sector, UAVs enable the execution of tasks that were once difficult, hazardous, or economically unfeasible. Applications include infrastructure inspection (e.g., bridges and power lines), agricultural monitoring for pest detection and crop health, and rapid, cost-effective package delivery services [1, 2, 3]. In the military context, UAVs were initially employed during World War II to evaluate sub-scale aircraft prototypes [4]. Today, they are extensively used for intelligence, surveillance, reconnaissance (ISR), and combat operations, serving as a critical component in defense strategies across the globe [5, 6, 7].

As the complexity of modern autonomous systems continues to grow, particularly those involving Remotely Piloted Aircraft Systems (RPAS) and multi-agent coordination, there is an increasing need for simulation platforms that can accurately represent physical dynamics while offering immersive, interactive 3D environments [8, 9, 10, 11]. Traditional dynamic simulation tools like MATLAB/Simulink and Hopsan

provide robust capabilities for dynamic modeling and control system development. However, they often lack native support for advanced visualization or seamless multi-agent interaction within a shared virtual space. Conversely, game engines such as Unity and Unreal Engine offer visually rich environments and extensive interactivity but typically require significant adaptation to support realistic dynamic simulations.

To address these limitations, this research explores software capable of improving this kind of simulation and presents an integrated simulation framework that combines the dynamic modeling capabilities of Hopsan, a high-performance object-oriented simulation platform utilizing transmission line modeling (TLM), with advanced rendering and interaction tools of Unreal Engine 5.3. This simulation architecture enables scalable, real-time multi-agent simulations, allowing users to study agent behaviors in complex environments with physical interactions and sensory feedback. The system is also designed to support human-machine interface (HMI) development, real-time data communication, and sensor emulation, making it a versatile platform for mission planning, operator training, and AI-driven control validation.

2 Software Platforms

During the initial phase of this study, a broad set of software tools was evaluated to address four essential criteria:

1. support for mathematical modeling of agents to ensure accurate physical dynamics;
2. high-fidelity 3D visualization capable of representing interactions among agents in a shared, interactive environment;
3. availability for academic use at no cost;
4. the ability to support real-time simulation performance; and
5. User Datagram Protocol (UDP) communication implementation capability.

The evaluation encompassed several widely adopted platforms, including FlightGear, AirSim, Unity 3D, Unreal Engine, Hopsan, and MATLAB/Simulink. Each tool was analyzed with respect to its capabilities, limitations, and overall suitability for multi-agent co-simulation. Key considerations included the level of modeling detail, scalability, ease of integration, and compatibility with external data interfaces.

The findings from this evaluation guided the selection of the simulation architecture and tools presented in this work. The subsequent sections outline the software exploration process and detail the implementation of the proposed integrated simulation framework that combines dynamic modeling and 3D visualization in a distributed, real-time environment.

2.1 Mathematical Modeling Software

- Hopsan: Open-source multi-domain system simulation tool. Hopsan uses built-in support for multi-core simulation and is built around the transmission line modeling (TLM) technique (or bi-directional delay lines). With this method, models can be automatically partitioned by introducing physically motivated time delays between components. Each model part can then be solved independently during each time step. This improves simulation performance, numerical robustness, and model scalability, makes the simulation very robust and fast, and provides a natural mechanism for parallel simulation. Hopsan is an object-oriented simulation platform written in C++. There is a strong focus on multi-core support and tool interoperability [12].
- MATLAB/Simulink: (with Aerospace Toolbox/Blockset) is a licensed academic software suite that combines MATLAB scripting and Simulink block-diagram modeling to implement fixed-wing aircraft flight dynamics. Users can visualize 3DoF/6DoF motion via MATLAB plots or leverage external simulators: open source FlightGear through Simulink interface blocks, or high-fidelity Unreal Engine using Simulation 3D blocks. Geometry can be imported, and cockpit-style instrumentation is available through dedicated Simulink blocks. External visualization is handled by FlightGear or Unreal, enabling realistic scene rendering while the core dynamic modeling remains within MATLAB/Simulink [13].

2.2 3D Visualization Software

- FlightGear: Open software that allows you to simulate an aircraft in the first person with very realistic dynamics. However, it is not possible to simulate several aircraft in the same instance and access the views through the same access window. An alternative is to use the tool's multiplayer; however, this requires a high computational effort and an infrastructure and network to distribute the process. The language of interaction is HTML [14].
- AirSim: Open software that allows multi-agent simulation of land vehicles and multi-rotors. As a software standard, it is possible to simulate quadcopter-type UAV and four-wheeled automobile models. Fixed-wing aeronautical models can also be implemented. It uses the Unreal engine as a viewer. The languages are Python and C++ [15].
- Unity 3D: Free software for academic research. It can develop the dynamics of fixed-wing aircraft and simulate multi-agents simultaneously within the same instance with access to different cameras on the agents. It can also build realistic scenarios with the Cesium Ion tool [16, 17]. The language is C$^\sharp$ [18].
- Unreal: Free software for academic research, like Unity 3D. It can develop the dynamics of fixed-wing aircraft and simulate multi-agents simultaneously within the same instance, with access to different cameras on the agents. It can also build realistic scenarios using tools such as Cesium Ion [16, 17] and ESRI ArcGIS [19]. The language is C++ and Blueprint [20].

Following a detailed assessment of available software tools against the defined simulation requirements, the implementation strategy advanced to a co-simulation framework combining Hopsan and Unreal Engine 5.3. Hopsan was selected for its robust capabilities in simulating the flight dynamics of fixed-wing aircraft using a transmission line modeling (TLM) approach, while Unreal Engine was chosen for its capacity to render high-fidelity, interactive 3D environments. Together, these platforms enable the seamless integration of dynamic modeling with immersive visualization.

Hopsan is a GPL-licensed, open-source, multi-domain simulation tool designed for high-performance academic research and development. It offers native support for parallel execution and multi-core simulation, making it well-suited for real-time dynamics. Unreal Engine 5.3, also available free of charge for academic use, provides advanced rendering, physics interaction, and interface customization capabilities through its Blueprint and C++ APIs. In the proposed architecture, Hopsan is responsible for generating the physical states of the simulated vehicle, while Unreal Engine interprets these states to drive graphical updates and interaction within the virtual environment. This division of responsibilities ensures both computational efficiency and visual realism, establishing a scalable platform for multi-agent simulation, HMI testing, and scenario-based analysis.

3 Framework Architecture

In the architecture used for this research, UDP socket libraries were developed in each application, enabling data transfer between the software entities involved. The part of the software developed to simulate the 3D representations was defined as the central point, since this part is responsible for receiving data from the dynamics, checking if there was an iteration with any part of the scenario, and returning the result of the iteration to the model. This part is also responsible for iterating with other peripherals coupled to it, such as an HMI, sensors, or simulated agents in other computational environments, shown in Figure 1.

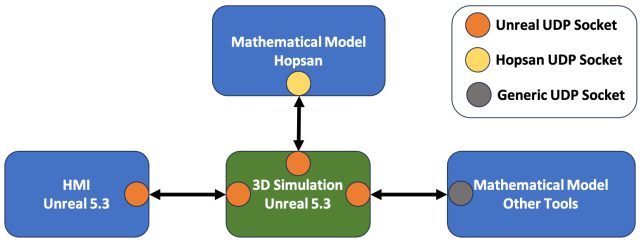

Figure 1: Framework Diagram.

The mathematical model embedded in Hopsan is responsible for calculating the dynamics of the vehicle or vehicles represented in the central 3D environment. Such dynamics can be enriched with details as necessary for performance evaluations during iterations in the 3D environment. For example, evaluating the performance of a hydraulic damping system for an iteration with a deformed surface or using fluid simulation tools to evaluate the iteration of a backhoe during excavation work [20].

The HMI, an optional part of the simulation framework, has as its main aspect the inclusion of a human being in the environment for iterations and evaluations of human factors within the simulation. This part can also be developed in the Unreal 5.3 environment and can use the same communication architecture as the main part.

As it is a multi-agent simulation environment, it is possible to couple more vehicle models represented by agents in the 3D environment or even sensor models in the preexisting dynamics into the simulation. This enables physical and visual interaction between them in the same virtual world.

The communication libraries developed will be demonstrated below, addressing their utilities, menus, and the possibility of modification for different applications.

4 Hopsan Library

The Hopsan UDP socket is composed of the library called *UnrealLib*. This library is used to connect Hopsan to Unreal software, based on the previous *FlightGearLib* [21]. The UDP communication is employed for sending and receiving data. The *UnrealReceive* and *UnrealSend* blocks handle the operation of this library and the reconfiguration of communication parameters in the Hopsan environment, Figure 2.

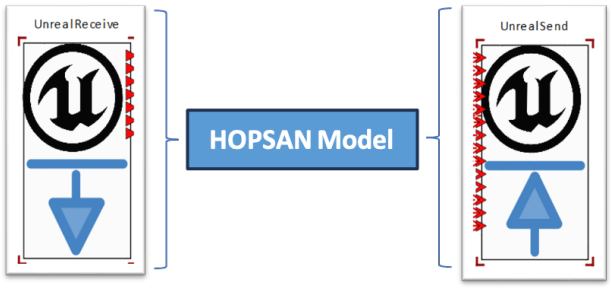

Figure 2: Hopsan library blocks.

4.1 *UnrealSend*

To employ the transmission block, we configured the sender's network parameters including IP address, port, and update frequency and applied a constant terrain altitude offset derived from the WGS-84 geoid to align the vehicle's altitude within the global reference frame. By default, the data stream comprises positioning, attitude, control surface, and anemometric variables; however, the configuration allows modification of both the data types and their transmission sequence. The block's configurable properties are illustrated in Figure 3.

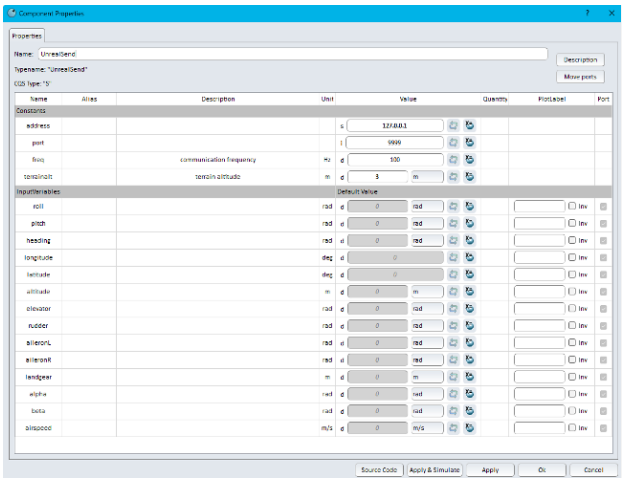

Figure 3: UnrealSend Properties.

It is possible to reconfigure the Send block, for this is necessary to change the files and recompile:

To modify the Send block, it is necessary to adjust the source files and recompile the *UnrealLib* module:

- .../Unreal_Lib/UnrealSend.hpp
- .../Unreal_Lib/UnrealSend.xml

This recompilation step allows the addition of custom data fields or alteration of the default message payload. It is imperative to maintain the exact sequence and data types of the structured variables, ensuring compatibility with the receiving software's parsing schema.

4.2 *UnrealReceive*

To configure the Receive block, the receiver's IP address and port must be specified on the Unreal side, while the reception frequency is managed within Unreal itself. By default, the block receives wind data, control surface commands, and terrain altitude, as illustrated in Figure 4. Its behavior and the sequence of received data can be adjusted via compile-time configuration.

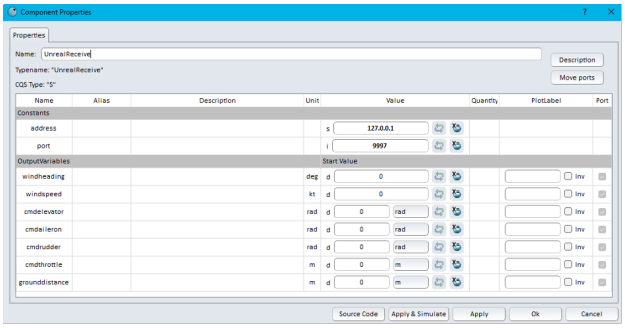

Figure 4: UnrealReceive Properties.

To reconfigure the Receive block, update the following source files and recompile the *UnrealLib* module:

- .../Unreal_Lib/UnrealReceive.hpp
- .../Unreal_Lib/UnrealReceive.xml

These modifications enable you to add additional data fields or alter the default bus message content. It is crucial that the order and data types of the structure's variables precisely match those expected.

5 Unreal Plugin

A dedicated UDP communication plugin was integrated into the Unreal Engine 5.3 simulation framework to enable real-time data exchange with external dynamic simulation tools. This plugin is based on an adapted version of the open-source project *UDPCommunication* for Unreal Engine 4, [22]. This library is designed specifically for use within Unreal's Blueprint visual scripting environment, allowing seamless integration into user workflows without requiring extensive C++ programming. It supports rapid prototyping and user-friendly configuration while maintaining low-latency and high-throughput data communication. It forms the core communication layer connecting the visualization environment in Unreal with the physics engine running in Hopsan.

5.1 *UDPSender* Module

The *UDPSender* module is responsible for transmitting simulation data from the Unreal environment to the external software Hopsan. It consists of two primary components, shown in Figure 5:

- *StartUDPSender*: Initializes the UDP socket and defines the local socket name, target IP address, and port used for transmission.
- *UDPDataSender*: Handles the structured transmission of data such as vehicle state, environmental conditions, and control inputs. The structure and order of transmitted variables must be consistent with the format expected by the receiving system.

Figure 5: UDPSender main blocks.

To modify the contents or structure of the transmitted data, users must manually edit and recompile the following files:

- .../Plugins/UDPCommunication/Source/...

- .../UDPCommunication/Classes/UDPData_Sender.h

This enables the addition of new data fields or customization of the default message payload. However, care must be taken to maintain structural compatibility with the receiving application.

5.2 *UDPReceiver* Module

The *UDPReceiver* module receives data from external simulation tools, Hopsan, and injects it into the Unreal Engine simulation environment. It comprises the following components, shown in Figure 6:

- *StartUDPReceiver*: Sets up the listening UDP socket, specifying the local socket name and listening port. The receiver uses the machine's own IP address as the default source address.
- *UDPDataReceiver*: Parses the incoming UDP packets into structured variables representing environmental conditions, actuator commands, or other simulation parameters.

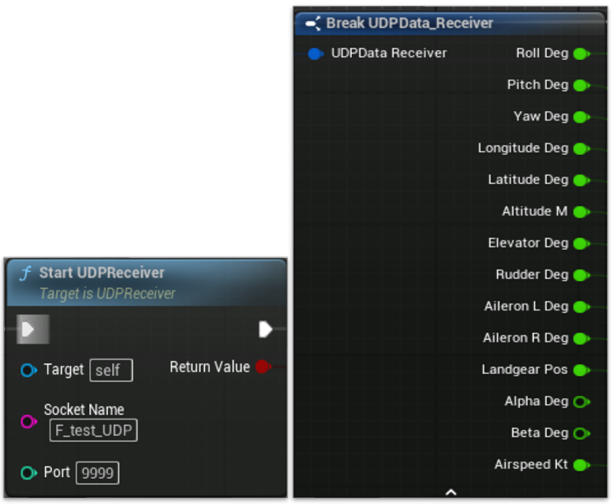

Figure 6: UDPReceiver main blocks.

Like the sender module, the receiver can be reconfigured by modifying and recompiling the source code in:

- .../Plugins/UDPCommunication/Source/...
- .../UDPCommunication/Classes/UDPData_Receiver.h

Maintaining consistency in the variable structure and type definitions between sender and receiver is essential to ensure correct data interpretation and avoid runtime errors.

6 Joystick Input in Unreal Environment

The joystick interface is configured via Unreal Engine's *Raw Input* plugin, which enables direct USB HID access to axis and button data. Devices are identified using their Vendor ID (VID) and Product ID (PID) hexadecimal values obtained from the operating system's device manager and registered in the Raw Input settings, shown in Figure 7.

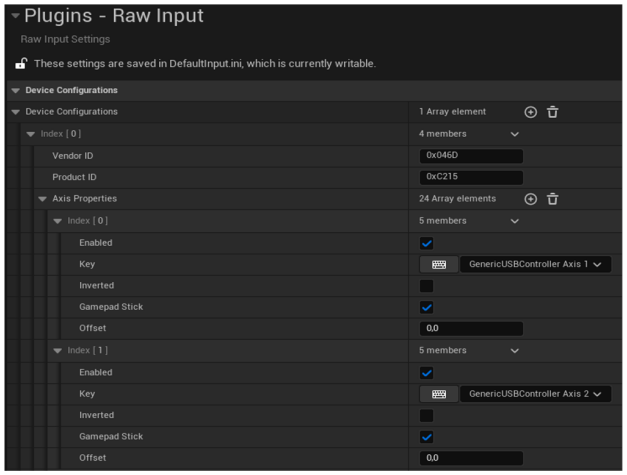

Figure 7: Raw input configuration.

Once recognized, each axis and button can be individually mapped, defining array indices and binding them to unreal action or axis events, with optional inversion and offset adjustments, Figure 8. This setup supports diverse USB joysticks natively.

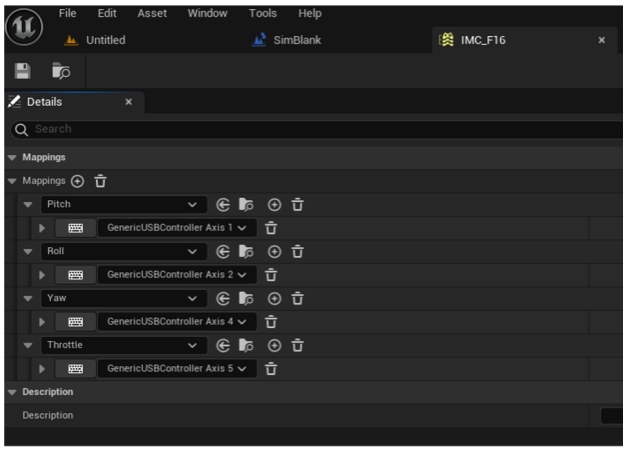

Figure 8: Control Mapping example.

7 3D Agent on Unreal Environment

To deploy a 3D agent in the Unreal environment, it is necessary to import a rigged 3D model with articulated components and define reference points, commonly referred to as joints or bones, on each movable part. These structural elements enable the rotation or translation of individual parts relative to other components or the global origin, facilitating accurate articulation within the simulation. For demonstration purposes, an F-16 aircraft model was integrated into the framework, complete with correctly mapped joints to control its movable surfaces, Figure 9. This skeletal setup follows Unreal Engine's standard process of importing skeletal meshes

(via FBX) with hierarchical bones for component animation and articulation. Moreover, this approach aligns with established practices in skeletal animation, where mesh deformation is governed by a connected hierarchy of joints and bones.

Figure 9: F-16 Bones in Unreal environment.

8 Results

The results of this study are organized into four distinct sections. The first section describes the distributed, multi-agent simulation architecture itself. The second section presents a validation scenario using a single F-16 aircraft to demonstrate the basic functionality of the framework. The third section extends this to a formation flight simulation involving two F-16s. Finally, the fourth section features a human-machine interaction scenario in which one aircraft is controlled via a joystick, and the second aircraft autonomously follows the same sequence of predefined waypoints.

8.1 Framework

The initial outcome of this work was the successful realization of the co-simulation architecture, combining two software environments to execute a unified simulation. Figure 10 illustrates this setup, displaying the Hopsan interface on the left and the Unreal Engine interface on the right. Although both applications were operated on the same physical machine for this demonstration, the framework supports distributed execution over a network, enabling the simulation to be partitioned across multiple computers.

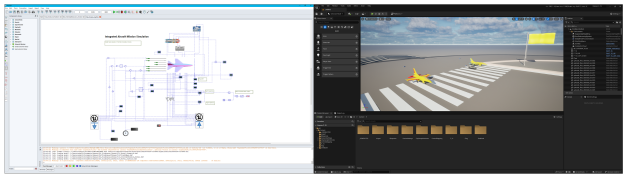

Figure 10: Framework program windows.

8.2 Single F-16 aircraft

Upon completion of the framework implementation, an F-16 aircraft model originally included in Hopsan was used as an initial validation case. This model incorporates waypoint-based guidance, an atmospheric model, propulsion system dynamics, actuators, and geoid-referenced altitude positioning. During the validation test, only Hopsan data was streamed to Unreal Engine, including attitude, navigation state, control surface deflections, and landing gear position. This integration and its visualization are depicted in Figure 11.

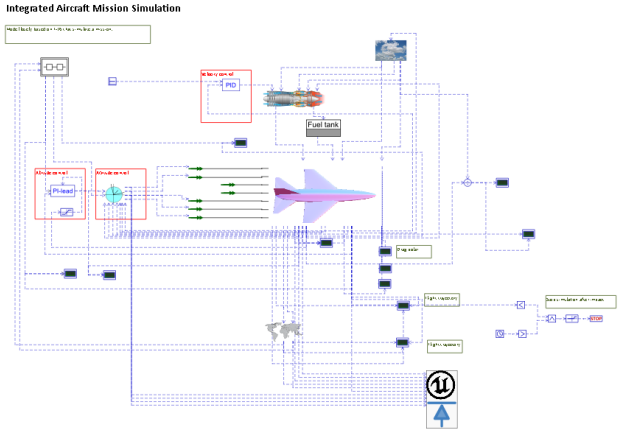

Figure 11: F-16 Hopsan example with UnrealSend.

Hopsan's F-16 implementation, which leverages its TLM-based architecture, has been extensively validated and is representative of complex flight dynamics simulation.

The simulation results rendered in Unreal Engine are shown in Figure 12. In this scenario, the landing gear is visibly extended because the corresponding deployment command was inactive within the Hopsan model. Consequently, the default gear position parameter was propagated throughout the simulation, resulting in the landing gear remaining fully lowered.

Figure 12: Flight of an F-16 in an Unreal environment.

This simulation was stored as a reference example demonstrating unidirectional data publishing from Hopsan to Unreal Engine. In this configuration, the Unreal environment passively consumes the data stream without transmitting any return signals or commands back to Hopsan. Simulation results, such as the position and altitude profiles from the F-16 interception mission shown in Figure 13, can also be plotted.

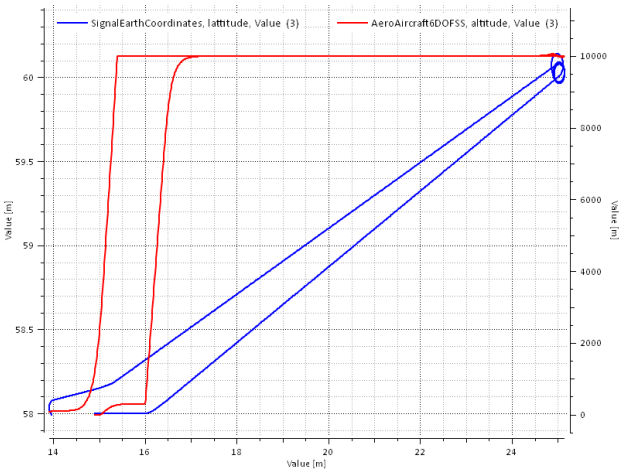

Figure 13: F-16 trajectory and altitude.

8.3 Two F-16s

Following the validation of single-agent data publishing, the framework was extended to a multi-agent configuration. Using the same F-16 model, each aircraft instance was assigned a dedicated UDP transmit and receive port within the Unreal environment, enabling independent bi-directional communication. This port-based architecture simplifies both message routing and potential future deployment across multiple machines. In the simulated scenario, all Unreal input variables, such as control commands and sensor data, were driven by Hopsan outputs, while manual control of one aircraft's landing gear was maintained, as illustrated in Figure14.

Figure 14: Flight of two F-16 in an Unreal environment.

This setup demonstrates flexible multi-agent coordination and modular network-based partitioning, enhancing scalability and maintainability in distributed simulation architectures.

8.4 Human in the loop

The final simulation scenario involves two aircraft operating within the same environment: one is piloted by an external agent via an HMI, while the other autonomously follows the same waypoints used in earlier experiments.

To incorporate a human operator into the simulation loop, an external input mode was introduced by integrating a USB joystick into the Unreal environment via the Raw Input plugin (see Section 6). A Logitech Extreme 3D Pro joystick was employed for this purpose, as depicted in Figure15. The joystick's Vendor ID (VID) and Product ID (PID) were registered to enable device recognition, with axis and button mappings configured within Unreal's input settings to correspond to flight control commands.

Figure 15: Joystick used in the simulation.

The joystick-generated data are received within Hopsan through the *UnrealReceive* block, Figure 16, which also integrates simulated sensor measurements such as altitude. In the simulation layout shown in Figure 14, the altitude reading is visualized as a blue elevation indicator beneath the aircraft, confirming the successful reception and integration of both control and sensor information into the Hopsan environment.

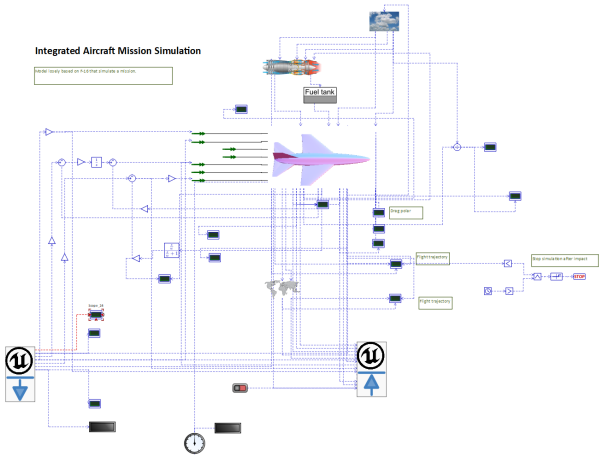

Figure 16: F-16 Hopsan example with UnrealSend and UnrealReceive.

9 Conclusion

This study successfully developed and validated a distributed multi-agent simulation framework that seamlessly integrates the dynamic modeling capabilities of Hopsan with the

high-fidelity visualization of Unreal Engine 5.3. The framework supports real-time bidirectional communication via a lightweight UDP-based plugin and accommodates human-in-the-loop interaction through USB Raw Input devices. Through sequential validation scenarios ranging from single-aircraft deployment to formation flight, and finally a joystick-controlled human-agent interaction, our results demonstrate the scalability, modularity, and performance of the platform suited to diverse research needs.

Key achievements include:

- Demonstration of synchronized multi-agent operation with independent UDP ports per agent.
- Effective handling of environmental physics (terrain, wind) and collision interactions.
- A flexible system architecture enabling possible distribution across multiple computers and support for scalable human-in-the-loop experiments.

Future work will focus on tighter coupling between Hopsan and Unreal Engine, especially using the built-in Unreal environment and fluid-dynamics capabilities, along with scalability studies involving larger agent ensembles. These directions align with established practices in distributed simulation to optimize parallel performance and agent synchronization.

This work lays a strong foundation for advanced UAS simulation, operator training, and AI-based mission planning, providing an open-source and extensible platform for ongoing development in distributed multi-agent environments.

Acknowledgement

The authors thank the Aeronautics Institute of Technology (ITA) and the Competence Center of Manufacturing (CCM) for their essential support and resources. We also gratefully acknowledge funding from the Brazilian Funding Authority for Studies and Projects (FINEP) through the Project ADS ("Cenários Futuros de Domínio Aéreo"), process no. 01.20.0195.00, National Council for Scientific and Technological Development (CNPq), Swedish-Brazilian Research and Innovation Centre (CISB), and Svenska Aeroplan Aktiebolaget (SAAB AB) which is crucial for the success of our research.

References

[1] Giacomo Rambaldi and Giovanni David. Unmanned aerial systems (uas) in agriculture: Regulations and good practices. Technical report, FAO & ITU, E-Agriculture in Action: Drones for Agriculture, Rome, Italy, June 2017. Published June 4, 2018 in FAO & ITU's "Drones for Agriculture" special edition :contentReference[oaicite:1]index=1.

[2] Yan Bo Huang, Steven J. Thomson, W. Clint Hoffmann, Yu Bin Lan, and Bradley K. Fritz. Development and prospect of unmanned aerial vehicle technologies for agricultural production management. *International Journal of Agricultural and Biological Engineering*, 6(3):1–10, 2013.

[3] He Luo, Yanqiu Niu, Moning Zhu, Xiaoxuan Hu, and Huawei Ma. Optimization of Pesticide Spraying Tasks via Multi-UAVs Using Genetic Algorithm. *Mathematical Problems in Engineering*, 2017.

[4] Glêvson Diniz Franco. Instrumentação e operacionalização de um sistema de ensaios em voo para VANTs (Vector-P). Master's thesis, Instituto Tecnológico de Aeronáutica, São José dos Campos - SP, 2009. 176f.

[5] DECEA. ICA 100-40: Aeronaves não tripuladas e o acesso ao espaço aéreo brasileiro. Technical report, DECEA, 2018.

[6] Federal Aviation Administration (FAA). Unmanned Aircraft Systems. `https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/unmanned_aircraft_systems.pdf`, 2023. Accessed: 2023-03-05.

[7] Congressional Research Service. Department of Defense Counter-Unmanned Aircraft Systems. Technical report, Congressional Research Service, 2020.

[8] Y. Chen. Path planning algorithm for logistics autonomous vehicles at cainiao stations based on multi-sensor data fusion. *PLoS One*, 20(5):e0321257, May 2025.

[9] Xiaoran Kong, Jianyong Yang, Xinghua Chai, and Yatong Zhou. An advantage duplex dueling multi-agent q-learning algorithm for multi-uav cooperative target search in unknown environments. *Simulation Modelling Practice and Theory*, 142:103118, 2025.

[10] Isuru Munasinghe, Asanka Perera, and Ravinesh C. Deo. A comprehensive review of uav-ugv collaboration: Advancements and challenges. *Journal of Sensor and Actuator Networks*, 13(6), 2024.

[11] Willson Amalraj Arokiasami, Prahlad Vadakkepat, Kay Chen Tan, and Dipti Srinivasan. Interoperable multi-agent framework for unmanned aerial/ground vehicles: towards robot autonomy. *Complex & Intelligent Systems*, 2(1):45–59, March 2016.

[12] Robert Braun, Petter Krus, Peter Nordin, and Björn Eriksson. Hopsan – an open-source tool for rapid modelling and simulation of fluid and mechatronic systems. *Proceedings of the ASME FPMC*, 2020. Describes Hopsan NG and TLM support.

[13] The MathWorks, Inc. Matlab version 9.14 (r2024b), 2024.

[14] The FlightGear Project. Flightgear flight simulator, 2024. Open-source flight simulator.

[15] Ratnesh Madaan, Nicholas Gyde, Sai Vemprala, Matthew Brown, Keiko Nagami, Tim Taubner, Eric Cristofalo, Davide Scaramuzza, Mac Schwager, and Ashish

Kapoor. Airsim drone racing lab. In *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, volume 123 of *Proceedings of Machine Learning Research*, pages 177–191, 2020.

[16] CesiumGS. Cesium for unreal. `https://github.com/CesiumGS/cesium-unreal`. Open-source Unreal Engine plugin, integrates Cesium Ion for 3D Tiles and real-world 3D geospatial streaming (Apache-2.0).

[17] CesiumGS. Cesium ion. `https://cesium.com/platform/cesium-ion/`. Cloud platform for streaming global high-resolution 3D content, integrates with Cesium for Unreal.

[18] Víctor Hugo Andaluz, Jorge S. Sánchez, Jonnathan I. Chamba, Paúl P. Romero, Fernando A. Chicaiza, José Varela, Washington X. Quevedo, Cristian Gallardo, and Luis F. Cepeda. Unity3d virtual animation of robots with coupled and uncoupled mechanism. In *AVR (1), Lecture Notes in Computer Science*, volume 9768 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 2016.

[19] Esri. Arcgis maps sdk for unreal engine. `https://developers.arcgis.com/unreal-engine/`. Official Unreal plugin providing real-world maps and 3D content via ArcGIS with support for 3D Tiles, OAuth 2.0.

[20] Epic Games. *Unreal Engine 5.3*, 2023. Version 5.3 release.

[21] Hopsan developers. Tutorial – Communicating with FlightGear. `https://hopsan.github.io/tutorials/tutorial_flightgear.pdf`, April 2022. Real-time data exchange between Hopsan (2.19.1+) and FlightGear via UDP sockets.

[22] Centre for Intelligent Systems, TalTech. Udpcommunication: A simple udp communication plugin for unreal engine. `https://github.com/is-centre/udp-ue4-plugin-win64`, 2025. MIT License; used for real-time communication in Unreal Engine via UDP.