



UNetyEmu: Unity-based simulator for aerial and non-aerial vehicles with integrated network emulation

Mauricio Rodriguez Cesen¹ , Ariel Góes de Castro¹ , Ibini A. Santana¹ ,
Ramon R. Fontes² , Fabricio R. Cesen³ , Christian Esteve Rothenberg¹

¹Faculdade de Engenharia Elétrica e Computação
Universidade Estadual de Campinas (UNICAMP) - SP - Brazil

²Leading Advanced Technologies Center of Excellence (LANCE)
Universidade Federal do Rio Grande do Norte (UFRN) - RN - Brazil

³Telefónica Research - Spain

{m272321, a272319, 206466}@dac.unicamp.br,
ramon.fontes@ufrn.br, fabricio.rodriguezcesen@telefonica.com,
chesteve@unicamp.br

Abstract. *This work presents UNetyEmu as a novel framework that enables realistic multi-vehicle experiments with aerial and non-aerial mobility alongside network emulation. By integrating Unity's 3D simulation and Mininet-WiFi's real-time network emulation, UNetyEmu allows researchers to conduct studies in smart city applications like 5G vehicular communication, edge computing, and drone coordination. Our framework is easily customizable and facilitates the evaluation of drone-related algorithms, such as obstacle avoidance and route planning, and studying network aspects, such as coverage and RSSI.*

1. Introduction

Unmanned Aerial Vehicles (UAVs), also known as drones, are rapidly transforming industries such as logistics, construction, and agriculture [Abderahman Rejeb and Treiblmaier 2023, AL-Dosari et al. 2023]. Their versatility extends to applications like disaster response, surveillance, and urban deliveries. However, ensuring safe and reliable operation becomes increasingly complex, particularly in terms of airspace management and wireless communications of multiple drones.

Most drones rely on either remote operators or autonomous navigation systems, both of which demand seamless coordination between onboard sensors (e.g., GPS, LiDAR, cameras) and robust wireless communication. However, drone networks face challenges such as mobility constraints, interference, or cybersecurity threats [Shafik 2023]. Traditional wireless network models do not fully capture complexities like mobility, computational capability, power consumption, and communication protocols [Hassija et al. 2021], underscoring the need for dedicated simulation tools.

It is essential to employ tools that accurately model and evaluate drone communications and operational performance. To this end, simulation tools enable researchers to assess environmental and operational risks while minimizing costs by not requiring extensive physical infrastructure. However, most existing drone simulators either fail to support multiple drones, oversimplify flight dynamics, or lack real-time network emulation, making them insufficient for realistic multi-drone scenarios.

To address the aforementioned aspects, we introduce UNetyEmu, a new framework that combines high-fidelity drone flight dynamics with real-time network emulation. Building on Unity’s 3D capabilities [Juliani et al. 2020], it offers a detailed virtual environment with realistic objects, including drones of various sizes, sensors, buildings, and urban infrastructure. Users can configure essential physical characteristics such as weight, battery capacity, and flight time, while also integrating algorithms for drone obstacle avoidance, path planning, and communication. The preliminary version includes a Proportional-Integral-Derivative (PID) controller and a logistics system that simulates package collection and delivery in future smart cities.

In parallel, we integrate UNetyEmu with Mininet-WiFi [Fontes et al. 2015] to enable real-time network emulation by executing network protocols directly on a virtualized Linux stack. Each drone is represented as a Docker container, allowing it to have distinct hardware capacities, coroutines, and processing power. These features ensure that networking experiments remain accurate and deployable, as the same configurations can be applied to real-world network hardware with minimal modifications. Combining Unity’s detailed physics-based simulation with Mininet-WiFi’s real-time network emulation, UNetyEmu enables realistic multi-drone experiments, supporting applications from path planning to large-scale drone coordination.

The rest of the paper is organized as follows: Section 2 reviews related work. Section 3 details the architecture of UNetyEmu. Section 4 presents the use cases and preliminary results. Section 5 shows the UNetyEmu repository and documentation. Finally, Section 6 concludes the paper with final remarks and directions for future work.

2. Related work

Experimental platforms, particularly drone simulators, have been investigated in the literature. Table 1 summarizes the related works in aspects such as: (i) 3D visually realistic environments; (ii) real drone flight dynamics, i.e., gravity, size, weight, load, battery, and sensors; (iii) support for run-time testing of algorithms, i.e., obstacle avoidance, path planning, and management; (iv) integration of non-aerial vehicles, and; (v) support not only simulation but also network emulation, i.e., base station to drones or drone-to-drone.

Table 1. Comparison between related works

Simulator	3D Realistic Environments	Real Dynamics in drone Flight	Run-time Algorithm Testing	Non-aerial Vehicles Integration	Network Emulation
UTSim [Al-Mousa et al. 2019]	–	–	✓	✗	✗
CoppeliaSim [Robotics 2024]	–	✓	✓	✓	✗
Gazebo [Koenig and Howard 2004]	✓	–	✓	✓	✗
AirSim [Shah et al. 2017]	✓	✓	✓	✓	✗
IoD-Sim [Grieco et al. 2021]	✗	–	✓	✗	–
RotorTM [Li et al. 2023]	✗	–	✓	✗	✗
gympybulletDrones [Panerati et al. 2021]	✓	✓	✓	✓	✗
FlightGoggles [Guerra et al. 2019]	✓	✓	✓	✓	✗
Webots [Michel 2004]	✓	✓	✓	✓	✗
Flightmare [Song et al. 2021]	✓	✓	✓	✓	✗
UNetyEmu (this work)	✓	✓	✓	✓	✓

✓ Supported Feature

✗ Not Supported Feature

– Partially Supported Feature

UTSim, for instance, an open-source simulator developed in Unity, allows the simulation of hundreds of drones in a single scenario and supports air traffic systems, navigation, and control. However, it lacks key dynamic elements such as realistic motion, position and rotation tracking, drone size and weight, supported load, battery life, and sensors integration. It currently lacks detailed documentation, and there is no mention of any support for non-aerial vehicles.

For its part, CoppeliaSim, a robotics simulation platform with ROS integration, is widely used to validate real-time motion control algorithms. However, creating a realistic scenario becomes a challenging task, and it has issues with the stability of the simulation as the number of drones in the scene increases. In the meantime, Gazebo is a widely used open-source simulator for robot testing, offering a customizable environment and support for control algorithms. However, it lacks a 3D model and scenario for drone research, so creating and modeling drone-related aspects from scratch is necessary.

Likewise, AirSim is popular in AI research and drone route planning tests. However, its LiDAR detection sensor is not fully realistic, and it is challenging to control multiple drones simultaneously. Additionally, IoD-Sim takes a different approach by focusing on drone network simulation and power consumption, using ns-3 [Riley and Henderson 2010] to evaluate communication protocols. However, it does not fully support network emulation and does not integrate drone flight dynamics.

Consequently, and to our best knowledge, none of the aforementioned simulators and others such as RotorTM, gypybulletDrones, FlightGoggles, Webots, and Flightmare possess native wireless network emulation. Therefore, UNetyEmu is the first to fill this gap by integrating Unity and Mininet-WiFi for high-fidelity flight simulation, real-time algorithm testing, and emulation of network aspects. As a result, our framework enables reliable and robust experiments on multi-drone communication scenarios in realistic environments.

3. UNetyEmu Architecture

To take advantage of all the network features available in Mininet-WiFi such as propagation models and signal strength/loss, as well as the compatibility and optimization features supported by Unity on Windows, UNetyEmu uses both Windows and Linux operating systems in this first release. The Figure 1 illustrates an overview of the UNetyEmu architecture, which mainly consists of two components: (i) Unity and (ii) Mininet-WiFi, as well as user inputs and outputs.

Input: User edits a JSON file to set all the initial features and components for each drone in the scene (see more details of this JSON file in the Appendix, Section A).

Unity: The core of our framework, where the sensors and drone dynamics are computed.

- **Initial Setup:** The user can configure the scene using Unity’s Hierarchy and Inspector toolbars (see more details in the Appendix, Section A). In them, it is possible to specify the characteristics of the 3D environment and the number of drones in the scene, editing the positions and orientations of the DronePads assigned for takeoff and landing.
- **Initialization:** At this point, the experiment begins, and the user may choose (or not) to use the network emulator. If the user chooses to do so, a coroutine is

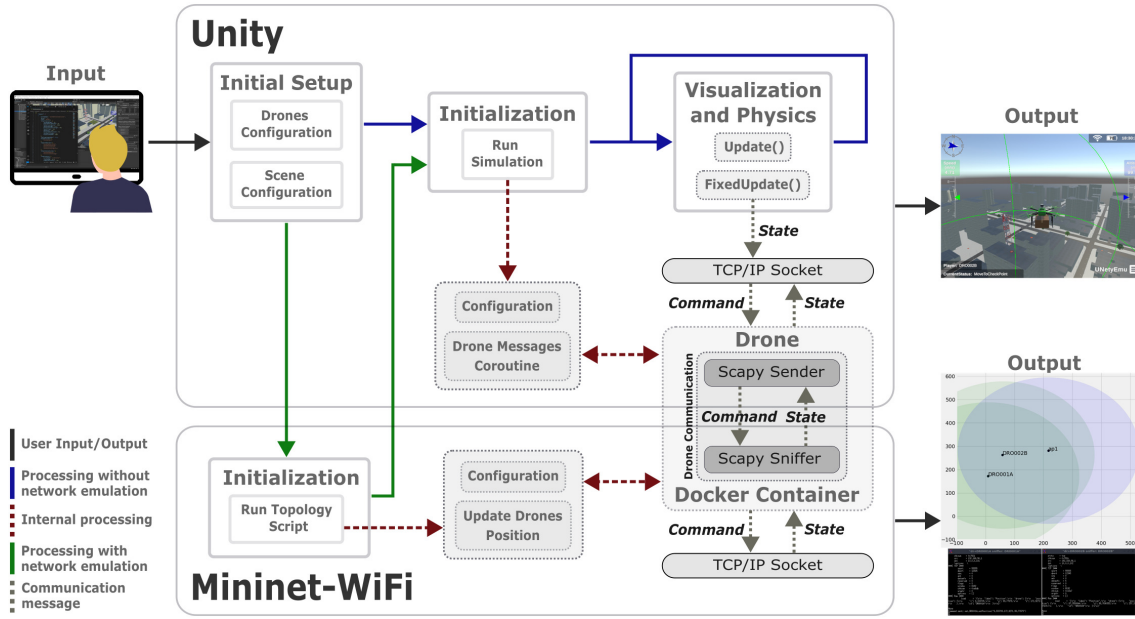


Figure 1. UNetyEmu Architecture

created in Unity to establish communication with Mininet-WiFi. By doing so, Unity receives the coverage value that the scene will use.

- **Visualization and Physics:** Unity provides two key functions: *Update()* and *FixedUpdate()*. The *Update()* is responsible for managing visual elements, such as rendering objects (e.g., trees, buildings, drones). On the other hand, the *FixedUpdate()* function is used to handle the physics aspects at specific frame rates, ensuring smooth and consistent movement of objects. Additionally, this function executes the drone control, management, and route planning algorithms.
- **Drone:** From a user perspective, drones are virtual nodes (Game Objects in Unity) managed collaboratively by Mininet-WiFi and Unity, appearing as a single unified entity. When Mininet-WiFi is used, we can exchange metadata (e.g., position, orientation) when extracting this information from the *FixedUpdate()* during the simulation, ensuring real-time synchronization between the flight dynamics and network emulation.

Mininet-WiFi: is the tool used for integration with network emulation. It offers accurate wireless modeling, SDN integration, and standard protocol support.

- **Initialization:** After the Initial Setup in Unity, the user just need to run the topology script (`mininet_topo.py`). We created custom JSON-formatted messages encoded/decoded in both sides of the communication as Scapy (see Listing 1). With this format, we can extend UNetyEmu to allow the exchange of different information (e.g., position, battery level, signal strength). Once it receives the first message from Unity with drones and base station positions', the containers are automatically created and they start listening for incoming messages to update the position of the drones in Mininet-WiFi.
- **Docker Container:** We consider a communication in which drones are connected to Base Stations (BSes) and can receive messages directly via their IP addresses and send object states (e.g., drone positions) back to Unity. Essentially, each

drone is a Docker Container with a wireless interface for communication. The integration between Mininet-WiFi and Unity is described in detail below:

- Message #1: The TCP/IP socket sends a command to a Scapy instance (e.g., the drone’s position);
- Message #2: Scapy processes the message received and encapsulates it in the payload of an IP packet, which is then sent to the drone;
- Message #3: A Scapy instance running on the drone receives the packet and forwards it to a TCP/IP socket in Mininet-WiFi, which interprets the message and converts it into a known instruction for the drone. The process of these messages is repeated throughout the experiment.

Unity Output: The user will be able to see in real time different indicators of the drone such as altitude, speed, orientation, task, battery level, and signal level.

Mininet-WiFi Output: The user will be able to observe a mirror of drone positions in Unity while observing the data packets arriving at each drone container.

Listing 1. Custom TCP/IP Packet

```
packet = IP(dst=destination_ip)/TCP(dport=destinaton_port)/<metadata>
```

4. Use Cases

UNetyEmu offers a versatile platform for assessing drone behavior, physics interactions, and network performance. To highlight different features of our tool, we present three scenarios with and without network emulation. The first scenario focuses on basic maneuvering and sensor functionality, the second extends to multi-drone simulation for tasks such as package delivery and obstacle avoidance, and the third scenario integrates network emulation on a drone package delivery.

4.1. First Scenario

Figure 2 illustrates a small environment with buildings, trees, and static objects. A blue DronePad marks the drone’s starting position and orientation, while the red DronePads indicate landing targets. This scenario uses only Unity (i.e., without network emulation) and showcases a 360-degree LiDAR sensor and a depth camera.

A PID controller stabilizes the drone, which follows a white sphere representing its target position. Users can adjust the target’s position with the `UJHK` keyboard keys and its altitude/rotation with the `arrow` keys.

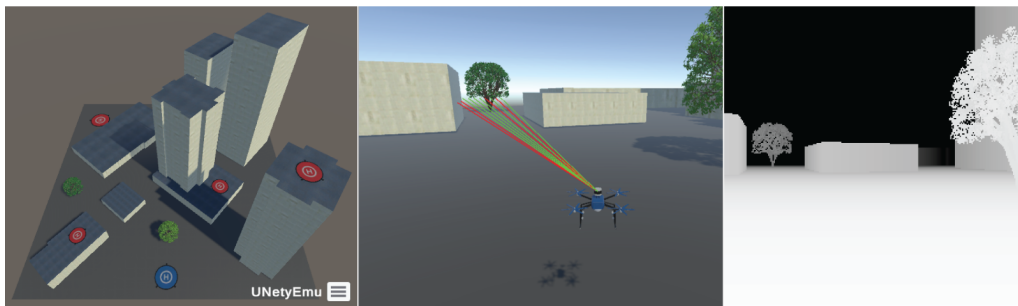


Figure 2. From left to right: Top view of the first scenario, LiDAR rays (red detection, green no detection), and depth camera output



Figure 3. Main camera tracking a drone, with real-time indicators. Left: A drone picks up a package. Right: A drone avoiding an obstacle

This scenario aims to understand sensor functionality, PID control, and drone maneuvering by landing on red DronePads. Replacing the keyboard keys, and with a trajectory and speed defined by the drone, think of the research possibilities in this simple scenario, such as obstacle avoidance and path tracking.

4.2. Second Scenario

The second scenario features four blue DronePads, where drones with different sensors, algorithms, and tasks will be instantiated. It uses only Unity and showcases UNetyEmu's capability to simultaneously simulate multiple drones with distinct behaviors.

The first two drones handle package deliveries by receiving locations for pickup and delivery, a predefined route, and speed/altitude constraints. They follow their assigned paths autonomously without using sensor feedback. The third drone, equipped with a depth camera, follows a predefined path while capturing images every 30 seconds. Finally, using a 360-degree LiDAR sensor, the fourth drone follows a trajectory while performing real-time obstacle avoidance by shifting right when it detects objects in any direction at a distance of less than 10 meters.

Figure 3 shows the first drone picking up a package and the last drone avoiding an obstacle. UNetyEmu provides real-time indicators such as signal strength, battery level, time, speed, altitude, drone name, and mission status. Battery consumption increases with motor usage, causing the drone to lose all motor power when the battery runs out. This scenario enables research on route planning, obstacle avoidance, dynamic object detection, depth image analysis, and battery optimization. Adding more drones is simple by editing in the Hierarchy and Inspector toolbar in Unity (see UNetyEmu documentation¹).

4.3. Third Scenario

Finally, the third scenario uses Unity and integrates network emulation into a package delivery simulation involving two drones in a city. It includes new static objects in the scene, such as benches, a car, and a base station. The experiment begins with the `mininet_topo.py` script running in the virtual machine. Unity then sends the base station and drone positions to Mininet-WiFi, which processes the data and returns the coverage radius. After Unity receives the coverage signal from Mininet-WiFi, the drones are notified of their missions and start delivering packages throughout the city.

¹<https://github.com/intrig-unicamp/UNetyEmu/wiki>

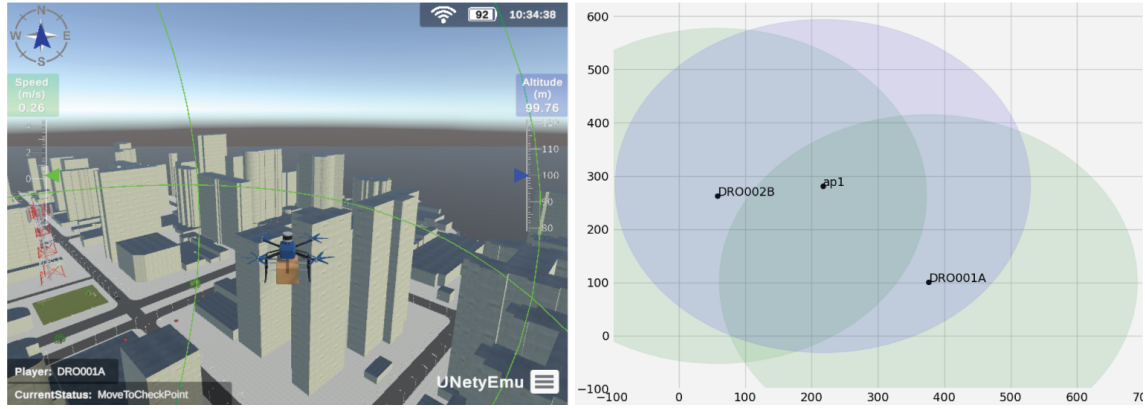


Figure 4. Package delivery simulation with network emulation. Left: Unity output. Right: Mininet-WiFi output

Figure 4 shows a drone delivering a package within its coverage area. At the same time, Mininet-WiFi emulates the positions of the Base Station and drones, in which each drone operates in an independent container, receiving real-time movement updates.

In both Unity and Mininet-WiFi, relevant information over time is stored. For instance, Figure 5 shows the altitude variations and the Received Signal Strength Indication (RSSI) for each drone over time. It can be noticed that before the second 25, despite flying at the same altitude, the drone DRO002B maintains a stronger and more stable signal than the other drone DRO001A, indicating that DRO002B was flying closer to the communication source (base station). Likewise, the opposite can be noted between the second 125 and 150, in which DRO001A flew closer to the base station. On the other hand, thanks to other stored data, it is possible to analyze the variations in battery consumption over time, depending on the type of task, such as taking off, moving to the target, landing, etc. (see more details of these results in the Appendix, Section B).

This scenario combines physical simulation with network emulation to enable realistic comparisons and consistent result validation. This reproducibility accelerates research while enhancing reliability and robustness. The work significantly contributes to validating algorithms for future smart cities, particularly in logistics, planning, and air traffic management within 5G-enabled environments.

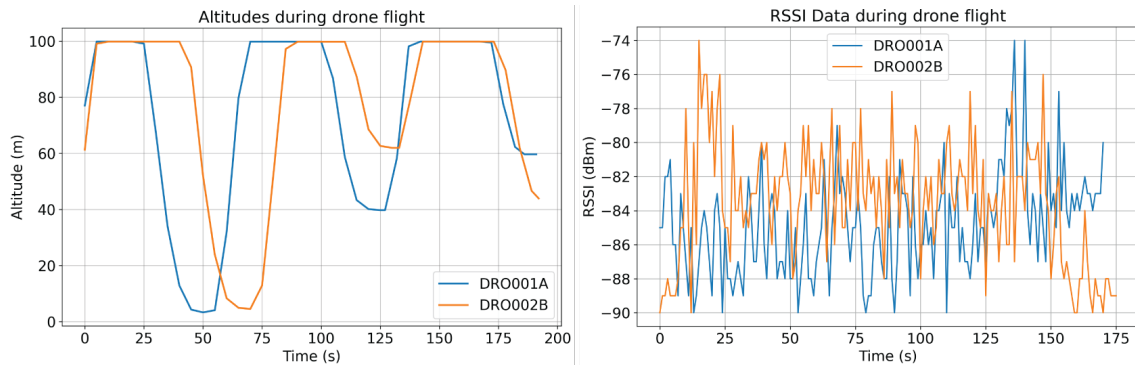


Figure 5. Left: Unity results on drone altitude over time. Right: Mininet-WiFi results on the RSSI of each drone over time

5. Documentation

UNetyEmu is an open-source project under the Apache License 2.0. To understand in more detail all the features that can be configured before and during the execution of each scenario, dive into the project repository and documentation:

- <https://github.com/intrig-unicamp/UNetyEmu>
- <https://github.com/intrig-unicamp/UNetyEmu/wiki>

We encourage readers to review all available UNetyEmu documentation, where the configuration of scenarios in the Unity Editor and the use of the initial configuration JSON file are explained in more detail. Additionally, the Basic Information and Dependencies pages show in detail the software and hardware requirements for a successful installation. We must emphasize that we are open to contributions, bug reports, discussions of current functionalities, and proposals for new features.



6. Conclusions and Future Work

This paper introduced UNetyEmu, a scalable framework that supports the implementation of control, obstacle avoidance, and route planning algorithms while integrating network emulation in diverse urban settings. UNetyEmu bridges the gap between high-fidelity mobility simulation and real-world network conditions, enabling smart city applications such as 5G communications and drone-based logistics. Our experiments demonstrate that it can simulate the behavior of multiple drones with varying physical and algorithmic traits under realistic conditions, including physical constraints. This makes it ideal for analyzing complex scenarios like urban air traffic and logistics in future smart cities.

While UNetyEmu integrates Unity and Mininet-WiFi to simulate vehicles with network emulation, several areas remain unexplored in the future. First, the full capabilities available when working together with Mininet-WiFi are still to be exploited. We are currently working on new scenarios where the loss of signal in the communication is considered. Another important aspect is to incorporate weather conditions, impacting drone mobility and network performance. Additionally, we want to facilitate the integration of AI and Python-based algorithms to open new research lines in optimization and multi-agent collaboration. In addition to the results shown in Figure 5 and Appendix B, we are working to export more statistics/indicators during the simulation of each scenario, e.g., packet delivery rate, task execution time, number of drones flying per minute, and collision detection, among others. Finally, we are working on incorporating UNetyEmu scenarios directly into Linux, avoiding dependence on Windows and a virtual machine.

Although each drone in UNetyEmu is represented as a Containernet object (Docker container) in Mininet-WiFi, we do not fully leverage Containernet's capabilities for heterogeneous computing. Therefore, we will explore drones with varying computational capacities, enabling more realistic simulations of drone networks for edge computing and dynamic task distribution. These advancements would further solidify UNetyEmu as a robust smart city drone network research platform.

Acknowledgments

This work was supported by Ericsson Telecomunicações Ltda., and by the São Paulo Research Foundation (FAPESP) , grant 2021/00199-8, CPE SMART-NESS . Also, this study was supported by CAPES/Brazil' postdoctoral grant, process 88887.005666/2024-00, and partially funded by CAPES/Brazil, finance code 001.

References

- Abderahman Rejeb, Karim Rejeb, S. J. S. and Treiblmaier, H. (2023). Drones for supply chain management and logistics: a review and research agenda. *International Journal of Logistics Research and Applications*, 26(6):708–731.
- AL-Dosari, K., Hunaiti, Z., and Balachandran, W. (2023). Systematic review on civilian drones in safety and security applications. *Drones*, 7(3).
- Al-Mousa, A., Sababha, B. H., Al-Madi, N., Barghouthi, A., and Younis, R. (2019). Ut-sim: A framework and simulator for uav air traffic integration, control, and communication. *International Journal of Advanced Robotic Systems*, 16(5):1729881419870937.
- Fontes, R. R., Afzal, S., Brito, S. H. B., Santos, M. A. S., and Rothenberg, C. E. (2015). Mininet-wifi: Emulating software-defined wireless networks. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 384–389.
- Grieco, G., Iacovelli, G., Boccadoro, P., and Grieco, L. A. (2021). Internet of drones simulator: Design, implementation, and performance evaluation. *IEEE Internet of Things Journal*, 10(2):1476–1498.
- Guerra, W., Murali, V., Ryou, G., and Karaman, S. (2019). Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Hassija, V., Chamola, V., Agrawal, A., Goyal, A., Luong, N. C., Niyato, D., Yu, F. R., and Guizani, M. (2021). Fast, reliable, and secure drone communication: A comprehensive survey. *IEEE Communications Surveys and Tutorials*, 23(4):2802–2832.
- Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., and Lange, D. (2020). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.
- Koenig, N. and Howard, A. (2004). 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)*, volume 3, pages 2149–2154. Ieee.
- Li, G., Liu, X., and Loianno, G. (2023). Rotortm: A flexible simulator for aerial transportation and manipulation. *IEEE Transactions on Robotics*.
- Michel, O. (2004). "cyberbotics ltd. webots: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*.
- Panerati, J., Zheng, H., Zhou, S., Xu, J., Prorok, A., and Schoellig, A. P. (2021). Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Riley, G. F. and Henderson, T. R. (2010). The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer.
- Robotics, C. (2024). Coppeliassim. Accessed: 2025-02-12.

Shafik, W. (2023). Cyber security perspectives in public spaces: drone case study. In *Handbook of research on cybersecurity risk in contemporary business systems*, pages 79–97. IGI Global.

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2017). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*.

Song, Y., Naji, S., Kaufmann, E., Loquercio, A., and Scaramuzza, D. (2021). Flightmare: A flexible quadrotor simulator. *2021 Conference on Robot Learning (CoRL)*.

Appendix

A. Drone and scene configuration

One of Unity's main features is its versatility in anchoring different Scripts to the same object. This allows developers to create several algorithms that fulfill specific functions, such as camera movement, updating indicators in real time, and, above all, the different features that can be included in a drone-type object.

Therefore, in all UNetyEmu scenarios, you will find an object in the Hierarchy toolbar named ObjectSetup, which is anchored to a Script called ObjectSetupScript. This script allows you to adjust, from the Inspector toolbar, several initial settings of the scene, such as adding the configuration JSON file, including different types of collisions to objects in the scene, associating drones with the same characteristics by groups, setting the number of drones to be instantiated and identifying the initial position of each drone in the scene.

Meanwhile, the JSON file allows customizing various drone characteristics, such as supported drone weight, battery level, obstacle avoidance, path planning algorithms, and communication capabilities. Figure 6 shows some components that can be added or excluded from the configuration file. Users can easily add or remove these parameters to adapt the simulation to their needs.

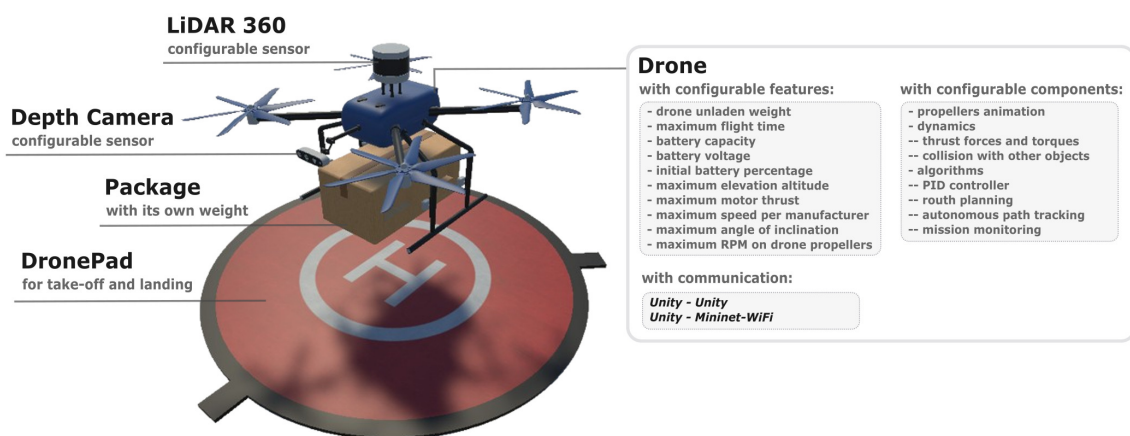


Figure 6. Drone characteristics and sensor types that are configurable from a separate JSON file for each scenario

An example of the configuration JSON file is shown in Listing 2. The main idea of this file is to group drones with the same characteristics to later instantiate them in a scene. The user will be able to include dozens of the same drone in the scene, identifying

only the name of the group in the Inspector and the DronePads in which they will be instantiated. You will also find references to the prefab name file corresponding to the 3D model of the drones available in UNetyEmu. Currently, there are three types of 3D drones, with and without sensors, and in two sizes. You can also identify the various algorithms that will be anchored to the drone, along with the physical characteristics of the sensors. As for the LiDAR sensor, you can alter the detection range, the horizontal and vertical ray levels, and the points per second detected. In addition, the depth camera sensor allows you to include features such as near clip plane, field of view, and pixel width and height. To better understand how the communication part works, please refer to the UNetyEmu documentation.

Listing 2. Example of a configuration JSON file

```
1 {
2   "players": [
3     {
4       "prefabName": "prefabDrone1CameraLidar",
5       "group": "A",
6       "type": "Drone",
7       "playerFeatures": {
8         "unladenWeight": 14,
9         "approxMaxFlightTime": 20,
10        "maxBatteryCapacity": 3000,
11        "batteryVoltage": 11.1,
12        "batteryStartPercentage": 100,
13        "maxAltitude": 80,
14        "maxThrust": 294,
15        "maxSpeedManufacturer": 30,
16        "maximumTiltAngle": 15,
17        "propellerMaxRPM": 7000
18      },
19      "addTargetGameObject": true,
20      "dynamicScripts": [
21        "DroneDynamics"
22      ],
23      "algorithmScripts": [
24        "DronePIDControlled",
25        "DroneRouthPlanning",
26        "DroneCurrentState",
27        "DroneInternalLogistics"
28      ],
29      "otherInternalScripts": [
30        "Drone1Animation",
31        "DroneJoinedToPackage"
32      ],
33      "lidarFeatures": {
34        "scriptName": "LidarSensor",
35        "lidarRange": 50.0,
36        "numRaysHorizontal": 360,
37        "numRaysVertical": 1,
38        "verticalFOV": 5.0,
39        "pointsPerSecond": 7000
40      },
41      "depthCameraFeatures": {
42        "scriptName": "DepthCamera",
43        "nearClipPlane": 0.3,
44        "farClipPlane": 60,
45        "fieldOfView": 60,
46        "pixelWidth": 256,
47        "pixelHeight": 256
48      },
49      "communicationFeatures": {
50        "scriptName": "DroneCommunication"
51      }
52    }
53  ]
54 }
```

It is important to emphasize that some of the 3D objects that are in the UNetyEmu scenes were designed from scratch, such as the drones, sensors, base station, drone pads, street lights, and benches; while, other objects such as buildings and trees were downloaded and edited using the free versions of BlenderKit and Blosm for Blender. Some textures were the authors' design, and others were downloaded and edited using the free version of ambientCG available online.

B. Drones battery consumption during a package delivery

Figure 7 shows the battery consumption of each drone during different tasks, such as *Land*, *MoveToPickupPackage*, and *TakeOff*, among others. In the image on the left, you can see the trajectory of each drone during the delivery of packages. Although there appears to be a collision in their trajectories, the drones perform their tasks at different times, so there is no collision between them. In the image on the right, it is possible to identify that the stages of landing and moving towards a target are the ones that consume the most battery power.

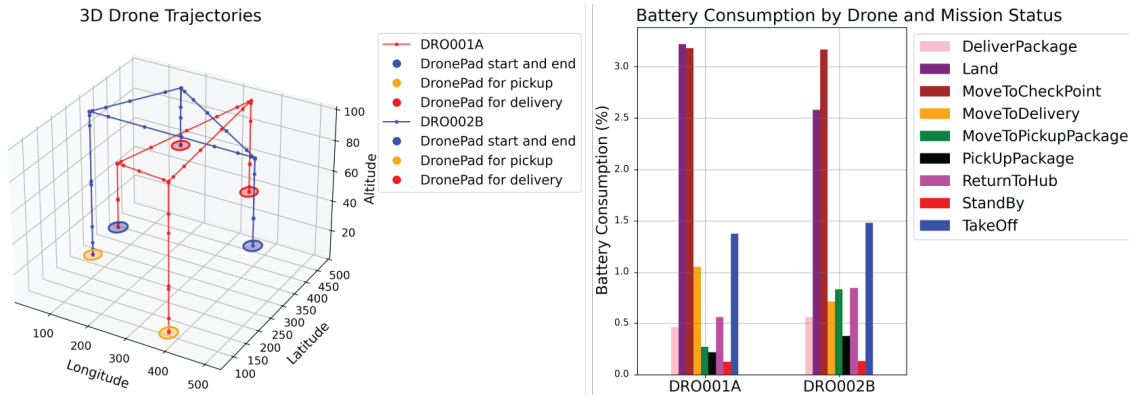


Figure 7. Left: 3D Trajectory of each drone during the delivery of packages. Right: Percentage of battery consumption by each task during the delivery of packages in the third scenario

It is important to emphasize that we use a simple battery consumption model that is affected by the elapsed time of the simulation and by the forces and torques applied in the drone dynamics. Thus, a strong movement that generates more force or displacement torque will consume more battery power.