# UNetyEmuROS: A Unity-Based Multi-Vehicle Simulator with Physically-Grounded Dynamics and ROS2 Sensor Integration

**Felipe Pavanello Capovilla**[1], **Mauricio Rodriguez Cesen**[2], **Christian Esteve Rothenberg**[2]

[1]Instituto de Computação (IC)
[2]Faculdade de Engenharia Elétrica e Computação (FEEC)
Universidade Estadual de Campinas (UNICAMP) - SP - Brazil

{f174411, m272321}@dac.unicamp.br, chesteve@unicamp.br

***Abstract.*** *We present UNetyEmuROS, a Unity-based multi-vehicle simulator that extends our previous work UNetyEmu with two key contributions: (i) a physically-grounded dynamics engine featuring per-motor forces, cascaded PID attitude control, and actuator disk energy modeling, where picking up a package physically alters thrust demand, inertia, and battery drain as emergent behavior; and (ii) a modular ROS2 sensor bridge publishing 360-LiDAR, RGB and depth camera, IMU, and GPS as standard* `sensor_msgs` *topics, each independently attachable to any vehicle. We validate both contributions in an urban scenario with heterogeneous drones and ground vehicles operating concurrently on package delivery tasks, object detection, and teleoperation through ROS.*

## 1. Introduction and Related Works

As unmanned aerial and ground vehicles (such as drones and autonomous cars) evolve toward large-scale deployment in urban areas, the need arises for simulation environments that combine physical fidelity, visual realism, and interoperability with external tools, such as artificial intelligence algorithms, especially for analyzing scenarios where multiple heterogeneous vehicles are performing different tasks simultaneously [Betti Sorbelli 2024, Makahleh et al. 2024, Munasinghe et al. 2024].

However, these requirements and use cases rarely coexist, as the available tools reflect a different set of priorities. Simulators like Gazebo [Koenig and Howard 2004] offer robust integration with ROS [Macenski et al. 2022] and detailed aerodynamic models for a single vehicle, but their 3D rendering systems are limited. AirSim [Shah et al. 2018], on the other hand, supports both quadcopters and ground vehicles in Unreal Engine, but its LiDAR detection sensors are not entirely realistic, and it is difficult to control multiple drones and cars at the same time; in addition, its official maintenance was discontinued by Microsoft. Also, Flightmare [Song et al. 2021] addresses the trade-off between speed and realism by decoupling a Unity [Juliani et al. 2020] rendering engine from an external physics backend, achieving unique simulation performance; however, this architecture requires the coordination of three separate processes, and its physical model focuses on the dynamics of independent quadcopters rather than interaction with payloads or ground agents, scenarios that will be fundamental for deployments of unmanned vehicles in urban areas. None of these platforms models payload pickup as a physical event that alters the drone's dynamics and power consumption, or provides a shared simulation environment for air-ground collaboration with standard ROS2 sensor interfaces.

Our previous work, UNetyEmu [Rodriguez et al. 2025a], partially addressed this challenge through a Unity-based multi-vehicle simulator with PID-controlled dynamics and multi-sensor support within the same graphical environment; it integrated the emulation of communication networks between vehicles and base stations [Rodriguez et al. 2025b], and served to validate route planning algorithms for 100 drones [Sow et al. 2026]. However, UNetyEmu had two important limitations. First, the drone flight model applied aggregate forces and torques to the center of mass, which limited physical representation at the rotor level and prevented integration with cascaded control architectures, that is, position (desired velocity), velocity (desired acceleration), and acceleration (desired pitch/roll). Second, sensor data remained confined to the Unity environment, with no standardized interface for external robotic pipelines. Additionally, although package pickup affected flight in reality, both propeller animation and the pickup mechanism operated as visual effects only, with no physical coupling to the dynamics.

In this work, we present UNetyEmuROS, a new version that extends our framework with two major improvements. Our first improvement, focused on drone flight, incorporates a quadrotor flight dynamics model based on individual rotors, accounting for motor forces, cascaded PID attitude control, waypoint tracking with a virtual target that moves along the planned trajectory ahead of the drone, and energy modeling using actuator disk theory. Additionally, we have improved the propeller rotation and the package pickup mechanism so that they are no longer executed as visual effects but by applying the forces derived from the new motion dynamics. This means that when a drone picks up a package, the increase in mass and the shift in the center of gravity alter the thrust demand, rotational inertia, and battery consumption as emergent consequences of the effect of gravity, and not as programmed disturbances. Our second contribution integrates a ROS2 sensor bridge that publishes 360-LiDAR, RGB and depth camera, IMU, and GPS as standard `sensor_msgs` ROS topics, each attachable to any vehicle (drone or car), enabling direct integration with external perception, planning, and control algorithms.

These improvements position UNetyEmuROS as a competitive simulation tool that combines physical realism, visual fidelity, and interoperability with any ROS2-based algorithm pipeline for the development and validation of heterogeneous multi-vehicle systems in urban environments.

The rest of this article is organized as follows: Section 2 details the system architecture. Section 3 presents the use case and preliminary results. Section 4 shows the UNetyEmuROS repository and documentation. Finally, Section 5 concludes the paper with final remarks and directions for future work.

## 2. System Architecture

All UNetyEmuROS vehicles are based on the same three-level modular architecture, as illustrated in Figure 1. The design principle is that a minimal `prefab` contains only the physical body; all behavioral (dynamics) and sensing capabilities (publisher/subscriber) are added as independent `MonoBehaviour` components that can be attached, removed, or swapped without modifying any other part of the vehicle.

Layer 1 provides the vehicle's physical model: for drones, forces and torques per motor are applied to Unity's `Rigidbody` via PhysX; for ground vehicles, an equivalent component controls wheel forces and steering. Layer 2 closes an optional internal con-

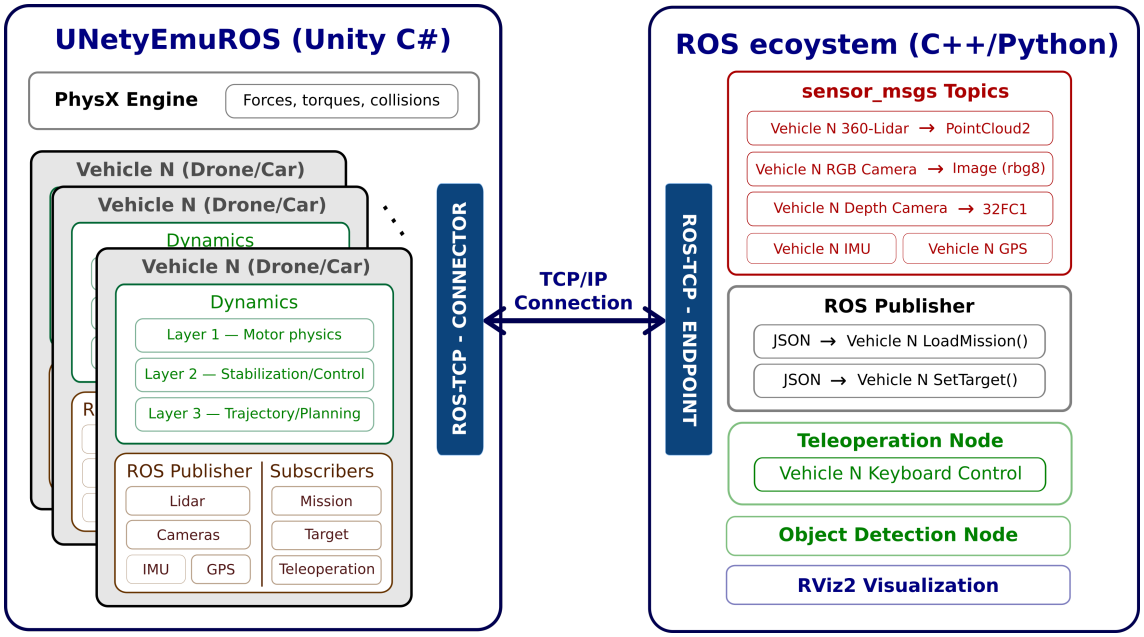


**Figure 1. Diagram of the UNetyEmuROS System Architecture**

trol loop, such as PID-based stabilization for drones, speed and course control for cars, which can be replaced by any custom controller without altering the dynamics. Layer 3 (currently available only for drones) handles trajectory tracking and mission execution, accepting waypoint sequences or mission plans focused on package delivery scenarios.

Regardless of these dynamic layers, any vehicle can be equipped with any of the sensors shown. Each operates autonomously, automatically registers its own topic, and publishes standard `sensor_msgs` messages via the ROS-TCP-Connector bridge. Sensors can be connected to a vehicle individually or all at once, turning every vehicle in the scene into a configurable ROS2 sensor source for external nodes, including additional functions for this demonstration such as mission loading, setting a new target, object detection, and teleoperation.

The following subsections detail the two main improvements introduced in UNetyEmuROS with respect to UNetyEmu [Rodriguez et al. 2025a]: the new physically-grounded vehicle dynamics (Section 2.1) and the ROS2 sensor integration (Section 2.2).

## 2.1. Drone Flight Dynamics Multi-Layer Architecture

### 2.1.1. Layer 1: Motor Physics

In contrast to UNetyEmu, which applied forces and torques to the drone's center of mass, UNetyEmuROS calculates and applies individual forces per motor directly to Unity's `Rigidbody` via PhysX in each `FixedUpdate` cycle, enabling a physics simulation at the rotor level. Each step performs: (i) calculation of total mass, including payload, (ii) quadratic aerodynamic drag, (iii) motor mixing, (iv) cross-coupling compensation, and (v) tilt angle limitation.

**Motor Mixing:** Total thrust is computed from the normalized throttle $\tau \in [-1,\, 1]$ and distributed across the motors using the standard X-configuration mixing algorithm for a quadcopter [Mueller and D'Andrea 2014]:

$$F_{\text{total}} = \text{clamp}\left(\frac{\tau + 1}{2},\, 0,\, 1\right) \cdot F'_{\text{max}} \tag{1}$$

where $F'_{\text{max}}$ is the maximum force per motor. Then, for each motor position (forward, backward, left, right), we determined the following:

$$\begin{aligned} \tilde{F}_{\text{FL}} &= 1 - p + r - y, \quad \tilde{F}_{\text{FR}} = 1 - p - r + y \\ \tilde{F}_{\text{BR}} &= 1 + p - r - y, \quad \tilde{F}_{\text{BL}} = 1 + p + r + y \end{aligned} \tag{2}$$

where $p,\ r,\ y \in [-1,\, 1]$ are the normalized pitch, roll, and yaw inputs. Therefore, forces are applied *at the exact position of each motor* via `AddForceAtPosition` and by distributing the total thrust $F_{\text{total}}$ among the motors. Aerodynamic drag, cross-coupling compensation, and tilt-angle limiting are applied additively in the same step [Pounds et al. 2010].

**Payload-Aware Mass:** We use Unity's native component, `FixedJoint`, to allow a payload to be attached to the drone. The total mass is automatically updated at each step of the physics simulation: $m_{\text{total}} = m_{\text{drone}} + m_{\text{package}}$. Also, to simulate the drone's limitation of carrying a payload greater than the manufacturer's specified limit, the motors are disabled when $m_{\text{package}} > m_{\text{max}}$. No controller parameters are adjusted; the resulting changes in thrust demand, rotational inertia, and power consumption are propagated as emergent behavior through the simulator's own gravitational forces.

### 2.1.2. Layer 2: Stabilization/Control

This layer handles the internal control loop between high-level attitude commands and the normalized signals used by Layer 1. This represents an improvement over the single-loop PID of our previous version, as it now implements three independent controllers operating at the PhysX cycle frequency [Mueller and D'Andrea 2014, Chovancová et al. 2014]:

- **Altitude** — PID on vertical speed with tilt compensation, preventing altitude loss during lateral maneuvers.
- **Pitch/Roll** — Rate-PD controller that combines the angle error (proportional) with the measured angular velocity (derivative).
- **Yaw** — Proportional yaw rate controller with seamless transition that synchronizes manual (direct modification of throttle, pitch, roll, and yaw variables) and autonomous (position and velocity control given a target) operating modes.

### 2.1.3. Layer 3: Trajectory/Planning

The position controller implements a cascaded PID architecture, enabling integration of position, velocity, and acceleration control [Chovancová et al. 2014, Faessler et al. 2017]. The waypoint follower applies the **Pure Pursuit** algorithm [Kishore et al. 2022]: instead

of guiding the drone toward each specific waypoint, a *virtual target* moves uniformly along the planned trajectory, staying ahead of the drone at a configurable lead distance. Furthermore, an effective lead distance is linearly reduced during the final approach to prevent overshooting the delivery point. Cruise, climb, and descent speeds per waypoint are interpolated along the active segment, resulting in smooth deceleration profiles without explicit speed planning.

$$\text{New Position} \begin{cases} e_y \xrightarrow{P} v_{y,\text{desired}} \xrightarrow{\text{PID}} \text{throttle command} \\ \mathbf{e}_{xz} \xrightarrow{\text{PID}_{\text{pos}}} \mathbf{v}_{\text{desired}} \xrightarrow{\text{PID}_{\text{vel}}} \mathbf{a}_{\text{desired}} \xrightarrow{\text{tilt}} \text{pitch/roll commands} \end{cases} \tag{3}$$

Meanwhile, the mission planner sequences several types of steps, such as `Takeoff`, `Cruise`, `Landing`, `PackagePickup`, `PackageDelivery`, `BatteryRecharge`, and `Idle` via a finite-state machine, coordinating the waypoint follower, position and velocity controls, the payload attachment mechanism, and the energy model. A ROS2 node can load a complete mission at runtime using `LoadMission()` or set individual position targets using `SetTarget()`, depending on which component(s) the drone has loaded at the start of the simulation.

### 2.1.4. Drone Energy Model

The power consumption per motor is calculated based on the thrust generated by the forces resulting from the drone's dynamic motion under the actuator disc theory [Mahony et al. 2012]:

$$P_i = \frac{F_i^{3/2}}{\sqrt{2 \cdot \rho \cdot A} \cdot \eta}, \qquad P_{\text{total}} = \sum_{i=1}^{N} P_i + P_0 \tag{4}$$

where $\rho$ is air density, $A = \pi r^2$ is the rotor disk area, $\eta$ is motor efficiency, $N$ is the number of motors, $P_0$ is a constant representing the energy consumed by the electronic components (flight computer, lights, sensors) regardless of the thrust generated by the motors, and $F_i$ is the force applied to each motor, as determined by the drone's flight dynamics. Also, the $P \propto F^{3/2}$ relation is superlinear; that is, a 25% increase in payload mass (for example, a 2-kg package on an 8-kg drone) results in a 40% increase in hover power consumption.

This asymmetry between outbound trips carrying cargo and return trips without cargo is directly relevant to energy-conscious route planning for urban delivery fleets [Dorling et al. 2017]. Battery status and estimated remaining flight time are continuously monitored and recorded for each vehicle instance in individual CSV files.

## 2.2. ROS2 Sensor/Control Integration

UNetyEmuROS incorporates a bidirectional communication bridge between Unity and ROS2 built on two components maintained by Unity: the *ROS-TCP-Connector* package [Unity Technologies 2026a], installed inside the Unity project; and the *ROS-TCP-Endpoint* node [Unity Technologies 2026b], compiled in a workspace and launched on

the ROS2 side. Once the endpoint is running, all sensor topics published by Unity become immediately visible via `ros2 topic list` and visualizable in RViz2.

Each ROS publisher/subscriber is implemented as a C# component that can be attached to any vehicle individually or all at once. On startup, each publisher/subscriber registers its own topic derived automatically from the name of the vehicle.

**Cameras (RGB and Depth):** The RGB camera renders the scene via an independent `RenderTexture` per vehicle using an `ARGB32` format and publishes a `sensor_msgs/Image` with `rgb8` encoding. Meanwhile, the depth camera uses `RFloat` format, publishing `32FC1` float images directly accessible by RViz2. Both cameras allocate independent render buffers per vehicle instance, preventing shared-state conflicts in multi-vehicle scenes.

**360-LiDAR:** simulates a rotating multi-channel sensor using Unity's **Job System** to execute all $R_h \times N_c$ raycasts in parallel, for instance, 360 horizontal rays $\times$ 16 channels, 45° vertical FOV, 30 m range, 10 scans/s. Only rays registering a collision are included in the output. Hit points are transformed to the sensor's local frame and remapped from Unity's left-handed axes to ROS: $(x_{\text{ROS}}, y_{\text{ROS}}, z_{\text{ROS}}) = (z_{\text{U}}, -x_{\text{U}}, y_{\text{U}})$. The result is published as `sensor_msgs/PointCloud2` with three `FLOAT32` fields and `is_dense=true`.

**IMU and GPS:** The IMU sensor publishes `sensor_msgs/Imu` synchronously with the Unity PhysX, carrying orientation, angular velocity, and linear acceleration estimated by first-order finite differentiation. In addition, the GPS sensor publishes `sensor_msgs/NavSatFix` at every physics step, mapping Unity world coordinates directly to geographic fields under a local flat-Earth approximation: $(x_{\text{U}}, z_{\text{U}}, y_{\text{U}}) \mapsto$ (longitude, latitude, altitude).

**Subscribers:** The subscriber components close the bidirectional bridge. `MissionReceiver` deserializes a JSON-encoded mission plan from a `std_msgs/String` topic triggering full autonomous execution. `TargetReceiver` accepts a `Float32MultiArray` carrying position, heading, and speed for single-target navigation. `DroneTeleoperation` maps $[\tau, p, r, y]$ for real-time drone remote control, and `CarTeleoperation` applies the same pattern to ground vehicles. In all cases, the topic name is derived automatically from the Unity `GameObject` name, giving each vehicle instance its own independent command channel.

## 3. Use Case

To validate both contributions, we designed an urban scenario with roads, buildings, and trees, in which three drones and four ground vehicles operate simultaneously. The drones differ in size, mass, and equipped components, as summarized in Table 1. Three ground vehicles remain parked, responding physically to collisions, while the fourth, equipped with GPS and IMU, is teleoperated from ROS2 via keyboard.

**Drone 1 (Autonomous delivery with 360-LiDAR):** This drone is a medium-sized blue quadcopter that receives a complete mission plan via `LoadMission()` from a ROS2 node, autonomously executing `PickingUpPackage`, `Takeoff`, `Cruise`, `Landing`, `PackageDelivery`, and then on the way back. Payload pickup is the key physical

**Table 1. Vehicle configurations in the demonstration scenario**

| | Size | Sensor<br>Unity → ROS2 | Control<br>ROS2 → Unity | ROS2 topic |
|---|---|---|---|---|
| **Drone 1** | Medium | 360-LiDAR | `LoadMission()` | `PointCloud2` |
| **Drone 2** | Large | Depth Camera | `SetTarget()` | `Image` (32FC1) |
| **Drone 3** | Small | RGB Camera | Keyboard teleop | `Image` (rgb8) |
| **Car 1** | Medium | GPS + IMU | Keyboard teleop | `NavSatFix, Imu` |

demonstration: when the drone picks up the package, the increase in total mass reduces vertical acceleration, and the altitude PID compensates by increasing the throttle as an emergent behavior of the physical motor. Simultaneously, the LiDAR sensor publishes a `PointCloud2` stream visualized live in RViz2, as shown in Figure 2.

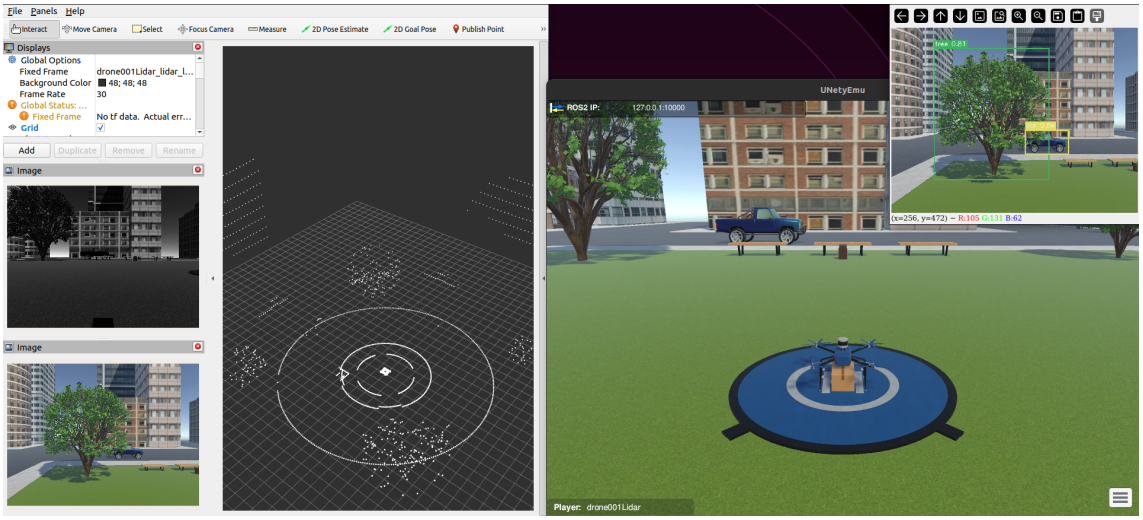


**Figure 2. UNetyEmuROS simulator in a simple urban environment involving ground vehicles and different types of drones operating simultaneously within the same Unity scene. On the left, the ROS sensor outputs in RViz2. On the right, the scene in Unity and the object detection using YOLO**

**Drone 2 (With depth camera and position control):** This drone is a large, heavier green quadcopter that receives individual position targets in the simulation via `SetTarget()` from a ROS2 terminal. Its greater mass results in a noticeably slower attitude response, which is perceived as a more gradual turn between targets. In addition, its depth camera streams `32FC1` frames in real time to RViz2 (Figure 2).

**Drone 3 (With RGB camera, object detection, and teleoperated):** This drone is a small, lightweight brown quadcopter that is keyboard-teleoperated via ROS2; its low mass yields the highest acceleration response. Its RGB camera stream is processed by a YOLOv8 [Jocher et al. 2023] node trained on a custom dataset of 470 simulated images of trees and ground vehicles, which processes the RGB stream. Utilizing the YOLOv8s pretrained weights over 100 epochs, the model achieved an mAP50 of 0.986 and a box loss of 0.38, demonstrating high localization accuracy despite the compact training set. The detection results are visualized in real-time as shown in Figure 2.

**Ground vehicles:** Three of the four ground vehicles have only their dynamics component loaded, remaining parked at different locations and responding physically to collisions. The fourth is a medium-sized blue truck equipped with GPS and IMU sensors that publish `sensor_msgs/NavSatFix` and `sensor_msgs/Imu` to ROS2, and is teleoperated in real-time via keyboard commands, demonstrating that the same sensor and control pipeline available for drones also applies to ground vehicles.

## 4. Documentation

The source code and the documentation of UNetyEmuROS, including all sensor publisher scripts, flight dynamics components, and ROS2 workspace configuration, are publicly available at:

- `https://github.com/intrig-unicamp/UNetyEmu`
- `https://github.com/intrig-unicamp/UNetyEmu/wiki`

## 5. Conclusions and Future Work

In this paper, we present UNetyEmuROS, an extension of UNetyEmu [Rodriguez et al. 2025a] with two contributions improving drone flight dynamics and interoperability in multi-vehicle simulation. The per-motor dynamics system, with cascaded PID control and actuator disk energy modeling, enables payload effects to emerge physically, altering thrust demand, rotational inertia, and battery consumption without manual disturbances. The ROS2 modular sensor bridge publishes 360-LiDAR, RGB and depth cameras, IMUs, and GPS as standard `sensor_msgs` topics, making each simulated vehicle a configurable sensor source for external ROS2 pipelines. Both contributions were validated in a heterogeneous urban scenario with three drones and four ground vehicles performing autonomous delivery, object detection, and teleoperation.

Beyond the demonstration, UNetyEmuROS serves as a research platform for obstacle avoidance, collision detection, payload-aware control, energy-optimized routing, and multi-agent air-ground coordination, among others. All testable through standard ROS2 interfaces without modifying the simulator's core. We also conducted a stress test on a Ryzen 9 9900X machine using a fleet of drones, each equipped with a 360-LiDAR sensor (the most computationally demanding component), each continuously streaming data from Unity to ROS2, showing stable operation at 30 FPS for 30 drones (40% CPU), peaking at 40 drones (6 FPS), with crashes beyond 50 units.

Future work will explore five main directions: (i) extending the energy model to include battery degradation; (ii) incorporating route re-planning based on object detection and LiDAR; (iii) scaling validation to larger fleets; (iv) enabling reinforcement learning for autonomous flight policy training; and (v) restoring integration with Mininet-WiFi to validate network emulation and vehicle-to-vehicle communication.

## Acknowledgments

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

## References

Betti Sorbelli, F. (2024). Uav-based delivery systems: A systematic review, current trends, and research challenges. *ACM J. Auton. Transport. Syst.*, 1(3).

Chovancová, A., Fico, T., Ľuboš Chovanec, and Hubinsk, P. (2014). Mathematical modelling and parameter identification of quadrotor (a survey). *Procedia Engineering*, 96:172–181. Modelling of Mechanical and Mechatronic Systems.

Dorling, K., Heinrichs, J., Messier, G. G., and Magierowski, S. (2017). Vehicle routing problems for drone delivery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(1):70–85.

Faessler, M., Falanga, D., and Scaramuzza, D. (2017). Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight. *IEEE Robotics and Automation Letters*, 2(2):476–482.

Jocher, G., Qiu, J., and Chaurasia, A. (2023). Ultralytics YOLO. `https://github.com/ultralytics/ultralytics`. Accessed: 2026-03-26.

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.

Kishore, K., Dalai, S., Jangir, Y., Singh, S., Rohan, M., Shashank, D., Katta, S. S. S., and Saha, S. K. (2022). 3d pure pursuit guidance of drones for autonomous precision landing. In *2022 13th Asian Control Conference (ASCC)*, pages 2218–2222.

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 vol.3.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074.

Mahony, R., Kumar, V., and Corke, P. (2012). Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics and Automation Magazine*, 19(3):20–32.

Makahleh, H. Y., Ferranti, E. J. S., and Dissanayake, D. (2024). Assessing the role of autonomous vehicles in urban areas: A systematic review of literature. *Future Transportation*, 4(2):321–348.

Mueller, M. W. and D'Andrea, R. (2014). Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 45–52.

Munasinghe, I., Perera, A., and Deo, R. C. (2024). A comprehensive review of uav-ugv collaboration: Advancements and challenges. *Journal of Sensor and Actuator Networks*, 13(6).

Pounds, P., Mahony, R., and Corke, P. (2010). Modelling and control of a large quadrotor robot. *Control Engineering Practice*, 18(7):691–699. Special Issue on Aerial Robotics.

Rodriguez, M., de Castro, A., Santana, I., Fontes, R., Rodriguez, F., and Rothenberg, C. (2025a). Unetyemu: Unity-based simulator for aerial and non-aerial vehicles with integrated network emulation. In *Companion Proceedings of the 43rd Brazilian Symposium on Computer Networks and Distributed Systems*, pages 100–111, Porto Alegre, RS, Brasil. SBC.

Rodriguez, M., de Castro, A. G., Fontes, R., Rodriguez, F., and Rothenberg, C. (2025b). An integrated framework for network emulation and multi-vehicle algorithm testing. In *Proceedings of the ACM SIGCOMM 2025 Posters and Demos*, ACM SIGCOMM Posters and Demos '25, page 167–169, New York, NY, USA. Association for Computing Machinery.

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2018). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Hutter, M. and Siegwart, R., editors, *Field and Service Robotics*, pages 621–635, Cham. Springer International Publishing.

Song, Y., Naji, S., Kaufmann, E., Loquercio, A., and Scaramuzza, D. (2021). Flightmare: A flexible quadrotor simulator. In Kober, J., Ramos, F., and Tomlin, C., editors, *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 1147–1157. PMLR.

Sow, A., Rodriguez, M., de Oliveira, F., Wzorek, M., de Leng, D., Tiger, M., Heintz, F., and Rothenberg, C. (2026). Multi uavs preflight planning in a shared and dynamic airspace. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026)*. Accepted for publication.

Unity Technologies (2026a). ROS-TCP-Connector. `https://github.com/Unity-Technologies/ROS-TCP-Connector`. Accessed: 2026-03-26.

Unity Technologies (2026b). ROS-TCP-Endpoint. `https://github.com/Unity-Technologies/ROS-TCP-Endpoint`. Accessed: 2026-03-26.