

TwinFlight: An Open-Source Sim-To-Real Pipeline for Autonomous Drones

Vijay Bajracharya
University of Delaware
Newark, Delaware, USA
vbajra@udel.edu

Zhuoyang Fu
University of Delaware
Newark, Delaware, USA
zhuoyang@udel.edu

Weisong Shi
University of Delaware
Newark, Delaware, USA
weisong@udel.edu

Abstract—Developing and testing autonomous drone software in simulation before real-world deployment is a common practice in the drone computing community. However, simulated environments rarely match the real operating site and the software tested in simulation often differs from what is deployed on real hardware. This results in critical failures that are only discovered for the first time during on-field missions. Existing approaches address these two problems in isolation causing defects that are rooted in the interaction between software and environment to persist. We present *TwinFlight*, a fully open-source sim-to-real pipeline built on PX4, ROS 2, and Gazebo that closes this gap by separating the application layer from the vehicle layer, allowing identical application-layer logic to run unchanged in simulation and on real hardware. This architecture is paired with a georeferenced site twin constructed entirely from publicly available LiDAR terrain data, aerial orthophotos, and OpenStreetMap building footprints, and a telemetry-mirroring digital twin that improves situational awareness during on-field missions. We evaluate *TwinFlight* across 60 flights in simulation and 15 flights in the real world, showing that the site twin achieves decimeter-level geometric accuracy and that real-flight cross-track error remains below 2 m throughout, with sub-meter RMSE across various mission types.

Index Terms—Autonomous drones, simulation-to-real, digital twin, georeferencing, PX4, ROS 2, Gazebo

I. INTRODUCTION

Simulation is widely regarded as an effective method for testing autonomous drone software due to its safety and repeatability [1]. It avoids licensing requirements, flight restrictions, and risk of damage to real infrastructure. However, reaching a meaningful simulation is more difficult than it appears. Simulated environments rarely match the real operating site and the code that is uploaded to the drone often differs from what was tested in simulation. These differences introduce integration problems that are only discovered during real hardware flights.

Existing work tends to address each problem separately. One line of work focuses on building realistic simulated environments that are faithful digital reconstructions of the real operating site [2] [3]. These works treat the digital twin reconstruction as the end product without carrying the same software through to real hardware. Another line of work focuses on the portability of the application logic [4] [5]. While some model the environment in 3D, these are typically confined to indoor or small-scale settings and are not georeferenced to real-world operating sites. The consequence

is that each approach can only expose the defects visible on its own side of the gap. As such, an integration failure that depends on both a software change and an environment mismatch survives both approaches.

To that end, we present *TwinFlight*, a fully open-source pipeline built on PX4 [6], ROS 2 [7], and Gazebo [8], that integrates these existing components into a single validated end-to-end methodology. It separates mission logic from environment variables, utilizes a georeferenced site twin and reuses the same simulator environment to provide a live digital twin of the drone that mirrors the vehicle’s pose inside the simulated world in real-time. *TwinFlight* has two defining properties:

Invariant application layer. The identical application layer consisting of navigation, mission execution, and any other on-board tasks the drone performs, runs in simulation and on the deployed drone without any modifications. Only configuration values that are inherently stage-specific, such as connection addresses are allowed to change. Everything that controls the behavior of the drone stays the same whether it is in simulation or the real world.

Site twin built entirely from public data. The simulated environment is constructed entirely from publicly available elevation data, aerial imagery, and map data. Therefore, a mission defined as GPS waypoints is usable without any changes both in simulation and the field. Furthermore, the construction of the simulated environment does not require dedicated aerial surveys or physical access to the site.

Our contribution is not a new component but the integration of existing open-source firmware with an open-source simulator and middleware alongside a site twin construction methodology, assembled into a single pipeline that is documented and empirically validated end to end. Concretely:

- 1) An open-source, two-stage pipeline built entirely from existing open-source firmware, simulator, and middleware components in which the application layer’s logic and structure are identical across both stages.
- 2) A reproducible methodology for constructing georeferenced site twins from publicly available terrain, imagery, and map data avoiding the need for custom aerial photogrammetry, flight permissions, or physical access to the site.
- 3) A flight study quantifying the fidelity gap between simulation and real flight.

The pipeline, including the application layer, the site-twin construction tooling, and the live twin bridge, is publicly available at <https://github.com/vijaymanbajracharya/twinflight>.

II. RELATED WORK

A. Drone Firmware

PX4 [6] is an open-source flight-control firmware that runs on dedicated autopilot hardware and implements low-level functions necessary to fly a drone. It performs functions such as sensor fusion, state estimation, attitude control, position control, missions, and fail-safe logic. It exposes a well-defined interface that allows navigation and decision-making software to run independently via either a ground station, a companion computer, or a remote control. This separation makes PX4 an attractive foundation for autonomous and intelligent drone missions.

ArduPilot [9] is the other major open-source autopilot firmware that predates PX4 and supports a broader range of vehicle types. Architecturally, they do not differ much as ArduPilot also reveals an interface for external commands and telemetry. The two open-source projects differ mainly in community convention. We build on PX4 for this work because of its existing native integration with ROS2 [7] and Gazebo [8] but the same pipeline design can be applied to either open-source firmware.

B. Drone Simulators

PX4 has native support for simulation-to-flight workflows [6], with Gazebo as the reference simulator [8]. It provides physics simulation such as rigid-body dynamics, collisions, and aerodynamics. It also provides sensor simulations such as cameras, IMUs, and GPS. All of these are configured inside a 3D world and communicates with PX4 over the same interface that it would for a real drone. This allows PX4 to perceive both the Gazebo simulated hardware and the real hardware as the same.

Beyond Gazebo, AirSim [10] is photorealistic simulator built using the Unreal Engine [11] aimed at perception-based machine learning research. Similarly, Flightmare [12] is another simulator build using Unity [13] that is focused more on large-scale dynamic simulations. In the same domain as our pipeline, the Pegasus Simulator [14] provides PX4 and ROS2 integration over NVIDIA Isaac Sim. These systems prioritize visual realism, and producing high-fidelity rendering of a specific real-world site which demands many hours of manual modeling, texturing, and tuning.

TwinFlight builds on the PX4 and Gazebo combination specifically because of its native, well-documented integration with ROS 2 [7]. ROS 2 is the middleware layer that connects the application layer to PX4, manages communication between onboard nodes, and provides the ecosystem of tools and packages that drone autonomy research depends on. This native three-way compatibility between PX4, Gazebo, and ROS 2 means that the same communication interface that runs in simulation carries over directly to real hardware without modification. Additionally, ROS 2 is the standard

middleware for robotics research meaning that integrating our pipeline into the broader robotics ecosystem is important for enabling future work that combines drones with other robotic platforms, shared perception pipelines, and multi-agent coordination frameworks that already exist within the ROS 2 community.

C. Environmental Site Twins

Reconstruction of real-world operating sites for UAV missions is an active area of research. Most existing approaches rely on dedicated capture of the site. UAVTwin [2] and EmbodiedSplat [3] construct site twins using 3D Gaussian Splatting for perception training and indoor navigation policies respectively, but require dedicated video capture of the target site to create the splat representation. This introduces time and equipment cost that a fast, low-cost twinning method aims to avoid. Valencia et al. [15] take a different approach, building ROS-Gazebo environments from free satellite data and validating trajectory prediction against real ArduPilot flights in mountainous terrain. This paper demonstrates that public data can support useful environment reconstruction without custom surveys. *TwinFlight* follows this public-data driven site twin construction methodology and extends it to fully geo-referenced terrain, imagery, and building reconstruction usable directly for GPS-based mission planning and pairs the resulting site twin with a software-invariant simulation-to-flight pipeline.

D. Behavior Frameworks

Several existing frameworks aim to standardize drone behavior across simulation and real environments. CrazySwarm [16] and CrazyChoir [17] target large swarms of Crazyflie nano-quadcopters, providing coordination between multiple vehicles across simulation and hardware. However, this framework is built around the Crazyflie platform specifically rather than a general PX4-based vehicle. Aerostack2 [4] offers platform agnostic mission and behavior definitions that target either a simulated or a real vehicle without modification. Aerostack2 is closest in terms of sharing the same goal of portable application logic across simulation and flight. Aerostack2 focuses more on platform generalization and application-layer abstraction whereas *TwinFlight*'s contribution is pairing an unmodified application stack with a fast, low-cost site twin, so that the simulated and real missions are not just running the same logic but also flying over a representation of the same physical environment.

III. SYSTEM ARCHITECTURE

A. Design Principle

TwinFlight is built around the idea of separating the application layer from the vehicle layer and forcing any communication between the two through PX4's standard uXRCE-DDS topic interface [18]. The application layer consists of the navigation logic, mission structure, and control nodes that determine what the drone does (Fig. 1). This layer is identical whether running in simulation or in the field with

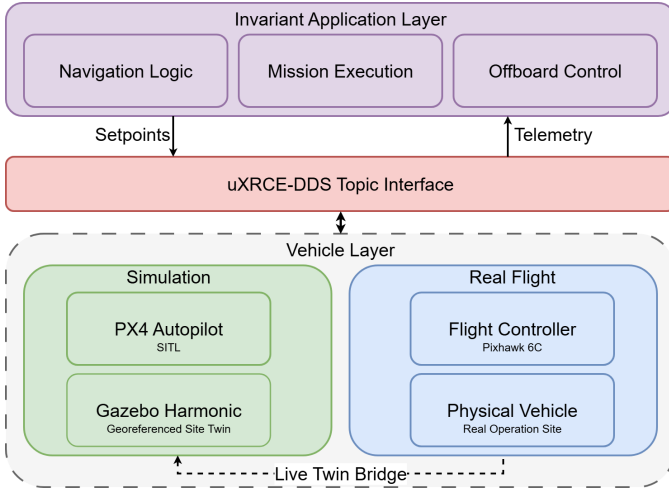


Fig. 1. *TwinFlight* architecture diagram

real hardware. The PX4 firmware itself is also constant across stages, running the same build in both simulation and the real hardware. The only aspect that changes is what the firmware is connected to. In simulation, PX4 runs against a simulated vehicle in Gazebo, with simulated sensors and motor response whereas in the field, identical firmware runs on the physical flight controller connected to real sensors and motors (Fig. 1). On the software side, only a handful of configuration values that are inherently stage-specific such as the transport address that the application layer connects to may change. These parameters live in an explicit configuration layer rather than being baked into the application logic itself.

This separation between the vehicle and the application allows *TwinFlight* to achieve its logic-invariance property, meaning, the application layer neither knows nor needs to know what it is talking to on the other side of the topic interface. Similarly, the firmware behaves identically regardless of whether it is connected to a simulated vehicle or the real vehicle.

B. Hardware Configuration

Our test platform was built using the Holybro X500 V2 development kit [19] which is a quadrotor frame widely used in the PX4 community. The flight controller is a Pixhawk 6C and is paired with a u-blox M10 GNSS GPS module and a 915 MHz SiK telemetry radio. An NVIDIA Jetson Orin NX is mounted onto the airframe as a companion computer and connected to the Pixhawk 6C over UART. The companion computer is responsible for hosting the application layer that runs identically in both simulation and real flight. Power is supplied by a 4S 5000 mAh 50C LiPo battery connected to a power distribution board that feeds the motors, electronic speed controllers, and dedicated voltage regulators for both the flight controller and the companion computer.

C. Software Configuration

The Jetson Orin NX runs Ubuntu 22.04 with ROS 2 Humble [7], which hosts the application layer including navigation

logic, offboard control nodes, and mission code. Communication between the application layer and PX4 passes through the Micro XRCE-DDS Agent [20], which exposes PX4’s internal publish-subscribe bus to a standard DDS network that ROS 2 nodes can subscribe to directly. The Micro XRCE-DDS Agent runs unmodified with the only difference being the connection changing from a UDP socket in simulation to a serial connection over UART in the real hardware. On the ROS 2 side, the application layer depends on the *px4_msgs* package [21], which mirrors PX4’s internal topic schema so that ROS 2 nodes can communicate with PX4 over the bridge agent.

IV. SITE TWIN

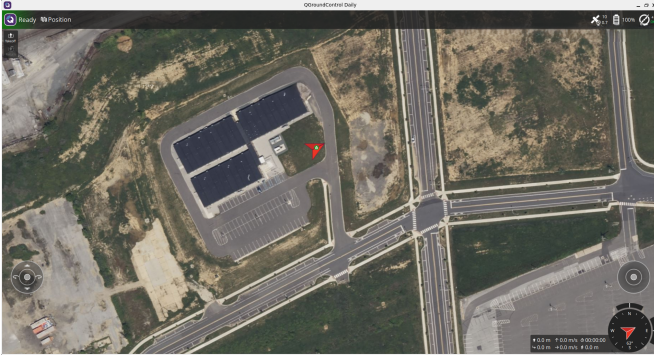
A. Data Sources

Rather than relying on dedicated photogrammetry and customized survey methods, for *TwinFlight*, we built the georeferenced site twin from existing public data. The bare-earth digital elevation model (DEM) and aerial imagery (orthophoto) were obtained from FirstMap Delaware [22] which is Delaware’s geographic information system data portal. Both datasets were queried using a common bounding box centered on the site so that elevation and imagery share the same spatial extent and coordinate reference system. Building footprints for structures within the operational area were obtained from OpenStreetMap [23]. A single point on the launch pad was surveyed and recorded in both WGS84 and the local state-plane projection. This point serves as the site datum that ties all elements of the simulated environment to actual real world coordinates.

B. Terrain and Building Reconstruction

The elevation raster is triangulated into a mesh, UV-unwrapped, and textured with the corresponding orthophoto (Fig. 3). It is then exported as a glTF Binary (.glb) for rendering in Gazebo. The terrain mesh is used for visual representation only and the physics collision is handled by a flat ground plane. This limitation stems from dartsim physics engine [24], which does not currently support non-convex mesh collision. We discuss this limitation and a potential fix in Section VI.

Building structures are reconstructed separately by extruding their OpenStreetMap footprints to a measured height and exporting them as Collada meshes with a flat material (Figure. 3). Building heights are measured without site survey by combining the bare-earth DEM used for the terrain, which captures ground-level elevation, and a first-return Digital Surface Model (DSM), which captures the elevation of the highest surface at each point including rooftops and above-ground structures. Sampling the DSM over each building’s footprint and subtracting the bare-earth DEM value results in the building’s height, requiring no manual measurement. Both the terrain and building assets are placed in the simulated world at their correct positions relative to the site (Fig. 2).



(a) Satellite imagery of operational area



(b) Site twin reconstruction in Gazebo

Fig. 2. Comparison between real operating site (a) and a render of Gazebo twin (b)

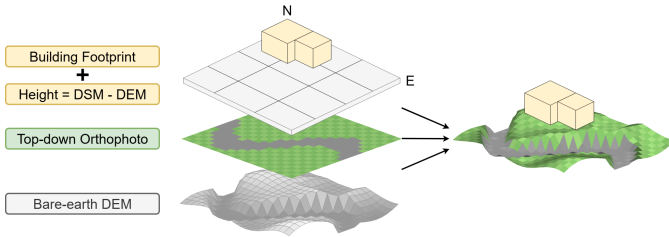


Fig. 3. Terrain and building reconstruction pipeline

C. Georeferencing

The simulated world’s coordinate origin is set to the surveyed anchor point in the real world. As such, PX4’s simulated GNSS reports true site coordinates. This allows GPS waypoint missions to be invariant whether it is in simulation or in the field. Additionally, any terrain-relative or proximity-based mission logic operates over a geographically consistent representation of the real operational site.

D. Live Twin Bridge

In addition to the site twin being used as a flight control validation tool before real flights, we also reused it to support a live telemetry mirroring bridge that visualizes real flights inside the simulated site twin in real time (Fig. 4). Vehicle position and attitude are transmitted from the drone to the ground station which is then applied to a separate “visual-only” model of the vehicle in the simulated world. A background task continuously holds the latest telemetry sample so that the mirrored model in Gazebo always reflects the vehicle’s current state without accumulating lag over a long flight. Ground level is auto-calibrated from the first GPS reading during the drone arming process.

The bridge connects to either the simulated vehicle during SITL sessions or the real vehicle during real on-field flights. This lets the same twin environment serve as both a flight control testing tool before real flights or an interactive live situational awareness display during real operations.

TABLE I
SITE TWIN TERRAIN ACCURACY

Metric	Full site	Core ($ \Delta \leq 2$ m)
Fraction of site	100%	97.0%
Mean Bias (m)	-0.207	-0.169
RMSE (m)	0.800	0.331
RMSE, bias-removed (m)	0.772	0.284
Median $ \Delta $ (m)	0.172	0.168
Max $ \Delta $ (m)	9.249	1.999

V. EVALUATION

We evaluate *TwinFlight* using a georeferenced mission path that remains unchanged between the simulation and the real flight.

A. Experimental Setup

We defined three benchmark missions, each expressed as a GPS waypoint plan used unchanged in simulation and the real flight: **M1** a 10 s position hold at 6 m above ground level; **M2** a 15 m $\times$ 15 m square circuit at 6 m; and **M3** a lawnmower pattern in a 15 m $\times$ 15 m area at 6 m;

Each mission was executed 20 times in simulation and 5 times in the field for a total of 60 simulation flights and 15 real flights. Simulated runs used PX4 SITL v1.17. Real flights used the reference vehicle from Section III-B running the identical PX4 release and application layer.

B. Site Twin Reconstruction Accuracy

We validate the accuracy of site twin reconstruction by comparing it against the United States Geological Survey (USGS) public LiDAR dataset. We obtained the USGS bare-earth DEM and the first return DSM through OpenTopography and compared it to the FirstMap Delaware DEM that was used to create the terrain mesh. Because the two datasets were collected by different organizations with different sensors, close agreement between the two indicates a faithful reconstruction.

Terrain. We computed cell-by-cell elevation difference between the two datasets, where each cell is a pixel in the elevation raster. We separated our findings into two categories:



Fig. 4. Simulation mirroring real-flight in site twin

TABLE II
CROSS-TRACK ERROR BY MISSION AND STAGE

Mission	Stage	Window	RMSE (m)	Mean (m)	Max (m)
M1	Sim	full	0.084	0.079	0.18
	Real	full	0.93	0.86	1.26
M2	Sim	full	0.19	0.16	0.72
		≥ 10 s	0.20	0.16	0.72
	Real	full	0.59	0.53	1.67
		≥ 10 s	0.58	0.52	1.67
M3	Sim	full	0.27	0.21	0.87
		≥ 20 s	0.30	0.24	0.87
	Real	full	0.81	0.69	1.99
		≥ 20 s	0.60	0.54	1.31

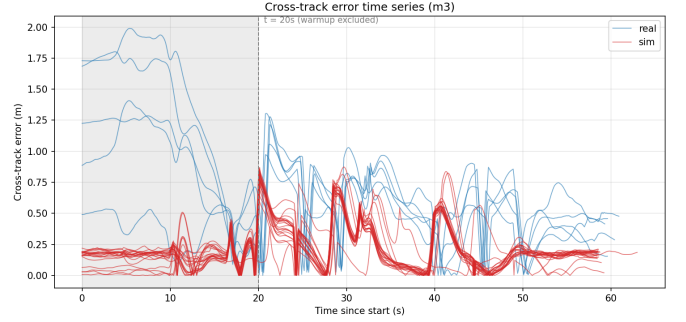


Fig. 5. Cross-track error for mission M3. Error over time for all runs, with the shaded region marking the excluded 20 s warm-up

full site, comprising every overlapped cell, and the core, comprising only cells that differ by less than 2 m.

From table I, we can see that the two datasets agree very closely across the full site with a median absolute difference ($|\Delta|$) of 0.172 m and a mean bias of -0.21 m meaning that our site twin reconstruction sits on average 0.21 m lower than the USGS dataset. 97% of the site that forms the core differs by less than 2 m and for this selection, the Root Mean Squared Error (RMSE) is 0.331 m. For the remaining 3% of cells, the RMSE is 0.8 m and the maximum difference is 9.25 m. These are spatially concentrated differences as a result of the time difference between the two dataset collections. Besides that, the two datasets have decimeter-level agreement which translates to an accurate reconstruction of the site twin.

Building Height. To measure the accuracy of the building height, we subtracted the USGS DEM from the USGS first return DSM which allowed us to sample the height of the building footprint. This resulted in a measurement of 4.3 m (interquartile range 4.18–4.40 m) for the one story building within the operational area, matching an in-person manual measurement.

C. Cross-Track Error

Table II reports the cross-track error for all three mission types in both simulation and real flight. For the missions

involving navigation (M2 and M3), we additionally report the cross-track error with the initial warm-up sequence excluded.

Across all three missions, the cross-track error for simulation was consistently lower than in real flight, as expected. Nevertheless, the maximum cross-track error in real flights never exceed 2 m in any mission and the RMSE and mean cross-track errors remained below 1 m for every mission in both simulation and real flight. As shown in Table II, excluding the initial seconds of each real flight reduced the cross-track error across all metrics, indicating elevated GPS uncertainty during takeoff and hovering. This behavior is further illustrated in Fig. 5 where the majority of the error is concentrated within the first 20 seconds of the mission.

With the site twin reproduction being within 0.3 m of the real site's geometry and the mission path being roughly within 0.6 m during navigation, a mission validated as safe and correct in simulation can be expected to remain safe and correct in the field as long as there is a sensible margin of error built into planning the autonomous missions.

The two results correspond to the two halves of the sim-to-real gap. The site twin's decimeter-level accuracy against independent public LiDAR data establishes that the simulated geometry is faithful to the real site, while close agreement between simulation and real-world in cross-track error for waypoint missions validates that the identical application logic

transfers to the field without behavioral changes.

VI. LIMITATIONS AND FUTURE WORK

TwinFlight presents a low-cost, practical path from simulation to real flight but it has several limitations:

Terrain collision. The terrain mesh in the current site twin is visual only. Simulated drones have collision physics with buildings but not with the terrain mesh extracted from the elevation raster. This limitation is due to the dartsim physics engine which does not support collision for non-convex meshes. One fix would be to switch to a physics engine that does support non-convex mesh collision but due to versioning constraints in our current Gazebo setup, we did not experiment with an alternative engine.

Single site and vehicle class. *TwinFlight* has so far been verified on a single site and one class of vehicle. We have not yet run experiments to determine how well the site-twin construction methodology transfers to other types of sites. Expanding the evaluation across multiple sites and vehicle configurations would help establish the generalizability of this approach.

Data currency and availability. *TwinFlight* relies entirely on publicly available elevation, imagery, and map data, which carries two related risks. First, public datasets are not updated in real time and may not reflect recent changes to the operational site. Second, the quality and availability of these datasets varies by region. High-resolution LiDAR and orthophoto coverage used in this work may not exist in areas without an established state GIS program. In areas such as these, the site twin would have to fallback to lower-fidelity public sources and coarser datasets.

Wind Simulation. The simulation does not model wind. Quantifying the effect of wind and modeling wind as part of the simulation would further close the gap between simulation and real flights.

VII. CONCLUSION

Simulation is widely recognized as a safe and cost-effective environment for developing autonomous drone software. However, discrepancies between simulated and real operating environments remain a persistent source of failure. *TwinFlight* addresses this issue on two fronts. First, it presents a logic-invariant application layer that runs without modification from simulation to real flight. Second, it provides a site twin construction methodology built entirely from public elevation, imagery, and map data making the simulated and real environment geographically consistent. Evaluations show that the site-twin reconstruction is faithful to the decimeter level, and that the cross-track deviation between simulated and real missions remains sub-meter on average. These results indicate that a mission validated as safe and correct in simulation reliably transfers to the field.

REFERENCES

- [1] C. S. Timperley, A. Afzal, D. S. Katz, J. M. Hernandez, and C. Le Goues, "Crashing simulated planes is cheap: Can simulation detect robotics bugs early?" in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2018, pp. 331–342.
- [2] J. Choi, D. Jung, Y. Lee, S. Eum, D. Manocha, and H. Kwon, "Uavtwin: Neural digital twins for uavs using gaussian splatting," *arXiv preprint arXiv:2504.02158*, 2025.
- [3] G. Chhablani, X. Ye, M. Z. Irshad, and Z. Kira, "Embodiedsplat: Personalized real-to-sim-to-real navigation with gaussian splats from a mobile device," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025, pp. 25 431–25 441.
- [4] M. Fernandez-Cortizas, M. Molina, P. Arias-Perez, R. Perez-Segui, D. Perez-Saura, and P. Campoy, "Aerostack2: A software framework for developing multi-robot aerial systems," *arXiv preprint arXiv:2303.18237*, 2023.
- [5] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio *et al.*, "Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight," *Science robotics*, vol. 7, no. 67, p. eabl6259, 2022.
- [6] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 6235–6240.
- [7] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science robotics*, vol. 7, no. 66, p. eabm6074, 2022.
- [8] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. Ieee, 2004, pp. 2149–2154.
- [9] ArduPilot, "ArduPilot," <https://ardupilot.org/>, accessed: 2026-06-27.
- [10] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics: Results of the 11th international conference*. Springer, 2017, pp. 621–635.
- [11] B. Karis and E. Games, "Real shading in unreal engine 4," *Proc. Physically Based Shading Theory Practice*, vol. 4, no. 3, p. 1, 2013.
- [12] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Conference on robot learning*. PMLR, 2021, pp. 1147–1157.
- [13] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar *et al.*, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.
- [14] M. Jacinto, J. Pinto, J. Patrikar, J. Keller, R. Cunha, S. Scherer, and A. Pascoal, "Pegasus simulator: An isaac sim framework for multiple aerial vehicles simulation," in *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2024, pp. 917–922.
- [15] E. Valencia, F. Toapanta, G. Oña, A. Carrillo, D. Aláez, E. Loyaga, W. Chamorro, P. Cruz, J. Abad, and P. Vandewalle, "An open-source uav digital twin framework: A case study on remote sensing in the andean mountains," *Journal of Intelligent & Robotic Systems*, vol. 111, no. 2, p. 71, 2025.
- [16] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "CrazySwarm: A large nano-quadcopter swarm," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3299–3304.
- [17] L. Pichierri, A. Testa, and G. Notarstefano, "Crazychoir: Flying swarms of crazyflie quadrotors in ros 2," *IEEE Robotics and Automation Letters*, vol. 8, no. 8, pp. 4713–4720, 2023.
- [18] PX4, "uXRCE-DDS (PX4-ROS 2/DDS Bridge)," https://docs.px4.io/main/en/middleware/uxrce_dds, accessed: 2026-06-27.
- [19] Holybro, "PX4 development kit X500 v2," <https://holybro.com/products/px4-development-kit-x500-v2>, accessed: 2026-06-27.
- [20] eProxima, "Micro XRCE-DDS Agent," <https://github.com/eProxima/Micro-XRCE-DDS-Agent>, accessed: 2026-06-27.
- [21] PX4, "px4_msgs: ROS 2 message and service definitions for the PX4 autopilot," https://github.com/PX4/px4_msgs, accessed: 2026-06-27.
- [22] The Location Intelligence (LI) team, "Delaware FirstMap," <https://firstmap.delaware.gov>, accessed: 2026-06-27.
- [23] OpenStreetMap, "OpenStreetMap," <https://www.openstreetmap.org>, accessed: 2026-06-27.
- [24] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. Karen Liu, "Dart: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.