FALCON-S: FIXED-WING AERODYNAMICS AND LEARNING CONTROL SUITE

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce FALCON-S, a modular and high-fidelity framework for learning and control of fixed-wing aerial vehicles operating in ground effect. In contrast to existing aerial platforms with simplified dynamics, FALCON-S incorporates full 6DoF simulation alongside detailed modeling of ground-effect aerodynamics, actuator dynamics, sensors and environmental disturbances. It offers a level of physical fidelity and modular component design that enables fine-grained manipulation and systematic analysis of low-altitude flight phenomena, capabilities rarely found in open-source or state-of-the-art simulation platforms. The framework includes both CPU and GPU simulation backends via Python and NVIDIA Warp, supporting high-throughput simulation across up to millions of parallel environments, which makes it suitable for reinforcement learning, sampling-based control algorithms, and large-scale evaluation. FALCON-S features a flexible architecture with interchangeable controllers, supporting optimal control, model-free and model-based RL, as well as a suite of flight control tasks such as altitude regulation and trajectory tracking. We include optional interfaces for validation and comparison through MATLAB/Simulink and X-Plane, making it compatible with both engineering workflows and commercial simulators. The framework is released as open-source to facilitate reproducibility and enable controlled benchmarking in realistic flight scenarios. Link to the code at *https://anonymous.4open.science/r/falcon-s-860E*.

1 INTRODUCTION

The design of intelligent control policies for aerial vehicles has advanced rapidly through the use of simulation-based learning. In particular, reinforcement learning (RL) methods have demonstrated remarkable capabilities in controlling complex robotic systems, from quadrotors (Kaufmann et al., 2023) and fixed-wing UAVs (Bøhn et al., 2019; De Marco et al., 2023) to satellites (El-Hariry et al., 2024) and legged robots (Chane-Sane et al., 2024). However, the practical deployment of RL policies on aerial vehicles remains limited by the gap between simulation and real-world dynamics, a gap often widened by the oversimplification of flight models, actuator behaviors, and environmental effects missing from existing simulators.

While simulation environments such as *JSBSim* (Berndt, 2004), *X-Plane* (Laminar Research, 2024), and *Flightmare* (Song et al., 2021) offer varying degrees of realism and extensibility, they typically lack features essential for research in modern RL and control: fine-grained aerodynamic modeling under phenomena like ground effect, GPU-accelerated simulation for sample efficiency, modular control integration, and compatibility with both model-based and model-free methods. Moreover, most existing platforms are tailored for either industry-focused pilot training or general-purpose physics simulation, offering limited flexibility for benchmarking learning algorithms across fidelity, dynamics, and control architecture variants.

In this work, we introduce a modular, physics-grounded, and RL-compatible simulation benchmark, focused on the control of fixed-wing vehicle, where ground effect, sensors and actuators realism, and full six-degree-of-freedom (6DoF) dynamics create a rich and challenging environment for learning and control. Our simulator provides:

(1) **Scalable dual-backend simulation engine:** We implement a GPU-accelerated physics pipeline using NVIDIA Warp alongside a CPU-compatible fallback, enabling high-throughput simulation for

both learning-based and classical control methods. The framework achieves a single-step simulation time of 0.0022 seconds at a scale of 10^6 parallel environments (on a desktop equipped with an NVIDIA RTX 4070-Ti SUPER GPU), which is a 100x speed-up w.r.t current state-of-the-art. Our simulator supports not only accelerated RL training but also efficient sampling for methods such as Model Predictive Path Integral (MPPI) and large-scale parallel evaluation.
(2) **Modular architecture for control and benchmarking:** The framework supports both reinforcement learning and optimal control methods (e.g., PPO, DreamerV3, LQR, MPPI), and includes a library of physically grounded flight tasks (e.g., altitude regulation, trajectory tracking) that can be easily extended or modified.
(3) **Comprehensive physics modeling:** Our simulation core incorporates aerodynamic models accounting for ground effect, wind, and atmospheric conditions, along with configurable sensor and actuator dynamics to support high-fidelity or ablation-based experiments.
(4) **Cross-platform validation interfaces:** We provide validation capabilities with both MATLAB/Simulink and X-Plane (for closed-loop testing in a high-fidelity commercial rendering engine), broadening applicability across academic and industrial settings.

This benchmark builds on a growing body of literature aiming to bridge control theory and deep learning in realistic flight settings. Notably, prior works have explored neural flight control under high-speed dynamics (Basescu et al., 2023), residual modeling of post-stall aerodynamics (Richards et al., 2021), and simulation-driven learning using platforms like *NeuralPlane* (Xue et al., 2024) and *QPlane* (Richter & Calix, 2021). Our contribution complements these by offering a *fully open and customizable framework* that blends scientific modeling (e.g., actuator dynamics, ground effect) with scalable learning infrastructure.
By supporting both classic and learning-based controllers within the same environment, we aim to foster reproducible research in control and learning under physically plausible dynamics. We demonstrate the use of both classical and learning-based controllers to illustrate the benchmarking capabilities of our framework. We envision this work as a step toward unifying optimal control and deep reinforcement learning in the context of aerial vehicle control, and as a building block for broader generalization across simulation-based autonomy research.

2 Related Work

Simulation of fixed-wing flight dynamics. Simulators such as *JSBSim*, *FlightGear*, and *X-Plane* have long supported fixed-wing aircraft modeling, but are primarily designed for pilot training or certification, and lack native support for reinforcement learning or scalable training. Recent research platforms such as QPlane (Richter & Calix, 2021) and NeuralPlane (Xue et al., 2024) address this limitation by exposing lightweight and configurable interfaces suitable for policy learning. QPlane wraps JSBSim for Gym-based RL experiments, while NeuralPlane introduces a parallel GPU-based pipeline for efficient large-scale simulation. However, both frameworks simplify critical aspects of flight dynamics, often using 3DoF or attitude-only models, with limited actuator fidelity and minimal environmental realism.
Flight control benchmarks and learning environments. While platforms like AirSim (Shah et al., 2017), Flightmare (Song et al., 2021), and RotorS (Furrer et al., 2016) have successfully advanced learning-based control for multirotor drones, fixed-wing benchmarks remain scarce due to the increased complexity of forward-flight dynamics, non-holonomic constraints, and sensitivity to external disturbances. Most existing learning environments focus on hover-capable vehicles, leaving limited support for lift-based platforms. Our work addresses this gap by introducing a unified simulation suite tailored to fixed-wing aircraft operating near the ground, combining realistic 6DoF dynamics with actuator and sensor models, ground effect, and wind disturbances. It supports both classical and learning-based controllers and achieves millisecond-scale single-step performance through GPU acceleration, enabling rigorous, scalable, and physically grounded benchmarking. **Combining classical control with deep learning.** There is increasing interest in combining optimal control methods with reinforcement learning (Berkenkamp et al., 2019; Liu et al., 2021). Works like Basescu et al. (Basescu et al., 2023) show how model predictive control can be extended with learned aerodynamic models to achieve aggressive post-stall landings. Similarly, residual RL and hybrid policy architectures have been used to improve control generalization while retaining safety guarantees. Our environment supports both classical baselines and learning-based controllers, enabling direct comparisons and hybrid control studies under consistent dynamics.
Modular, accelerated simulators for RL. Efficient learning requires simulators that are both

Table 1: Comparison of our platform with existing aircraft simulation frameworks. Our system combines realistic near-ground fixed-wing aerodynamics with modular flight tasks and supports advanced controllers in a reinforcement learning context, with rich sensors and actuators modeling, while enabling expandability for sim-to-real transfer. [✓] fully supported,[*] partially or optionally supported, [–] not supported.

Feature	Ours	NeuralPlane	QPlane	JSBSim	XPlane
Open-source	✓	✓	✓	✓	–
Physics-based FDM	✓(WIG, 6DoF)	✓(fixed-wing only)	✓(JSBSim/XPlane)	✓	✓
Ground Effect Model	✓(semi-empirical)	–	* (depends on JSBSim)	*	✓
GPU Acceleration	✓(Warp)	✓(PyTorch)	–	–	–
Multi-agent Support	–	✓	✓	*	* (via UDP)
Multiple Flight Tasks	✓	✓	✓	*	*
Controller Support	✓	✓	✓	*	*
Realism	High	Medium	High (if X-Plane)	High	High
Visualization Tools	✓	*	*	* (via FlightGear)	✓
Sim-to-Real Ready	*	*	*	✓	✓

fast and customizable. GPU-accelerated simulators like WarpDrive (Pan et al., 2021) and Isaac-Gym (Makoviychuk et al., 2021) have become increasingly popular in robotics research, but few have targeted flight vehicles. Flightmare (Song et al., 2021) provides GPU acceleration via Unity, yet focuses on quadrotor dynamics. Our Warp-based simulator offers domain-specific GPU acceleration for fixed-wing vehicles with detailed aerodynamics, supporting large-scale training without compromising physical realism.

To contextualize our contribution, Table 1 presents a detailed comparison between our simulation platform and several prominent aircraft simulation frameworks, including NeuralPlane (Xue et al., 2024), QPlane (Richter & Calix, 2021), JSBSim (Berndt, 2004) and XPlane (Laminar Research, 2024). While prior systems offer valuable capabilities, such as high-fidelity physics engines, Gym-compatible RL integration, or large-scale parallelism, most fall short in supporting near-ground aerodynamic effects or unified, extensible control pipelines. In contrast, our platform combines realistic 6-DoF flight dynamics with explicit ground effect modeling, modular control integration (classical and learning-based), precise actuator and sensors modeling and support for advanced aerodynamic modeling and realistic disturbance injection, like wind turbulence, atmospheric pressure (for high altitude flight conditions) and simplified computation of aerodynamics coefficients with OpenVSP (NASA OpenVSP Team, 2025).

3 PRELIMINARIES

We consider the control of a rigid fixed-wing vehicle flying in proximity to the ground, modeled as a six-degrees-of-freedom (6DoF) system with coupled translational and rotational dynamics. The vehicle is subject to forces from gravity, aerodynamics, and propulsion, and its motion is described in the body frame. The state vector $\mathbf{x} \in \mathbb{R}^9 \times S^3$ (or $\mathbf{x} \in \mathbb{R}^{12}$) comprises the position $\mathbf{p} \in \mathbb{R}^3$, orientation (represented as an unit quaternion $\mathbf{q} \in S^3 = \{\mathbf{q} \in \mathbb{H} : ||\mathbf{q}|| = 1\}$ or Euler angles $(\phi, \theta, \psi) \in \mathbb{R}^3$), linear velocity $\mathbf{v} \in \mathbb{R}^3$, and angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$. Control inputs include throttle and actuator deflections for the elevator, rudder, and ailerons. The equations of motion follow Newton-Euler rigid body dynamics:

$$m\dot{\mathbf{v}} = \mathbf{F}_g + \mathbf{F}_a + \mathbf{F}_t - \boldsymbol{\omega} \times m\mathbf{v}, \tag{1}$$

$$\mathbf{J}\dot{\boldsymbol{\omega}} = \boldsymbol{\tau}_a + \boldsymbol{\tau}_t - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}, \tag{2}$$

where m is the vehicle mass, $\mathbf{J}$ is the inertia tensor, $\mathbf{F}_g$ is the gravitational force, $\mathbf{F}_a$ and $\boldsymbol{\tau}_a$ are aerodynamic forces and moments, and $\mathbf{F}_t$ and $\boldsymbol{\tau}_t$ are thrust-generated force and moment vectors. We assume constant mass and neglect gyroscopic effects. Each vehicle is modeled as a rigid body with a body-fixed frame $\{b\}$ rigidly attached at the centre of mass, and motion is described relative to an inertial north–east–down (NED) frame $\{I\}$.

Aerodynamic forces and moments are computed using semi-empirical models based on the vehicle's angle of attack α, sideslip β, Reynolds number Re and control surface deflections. Lift, drag, and side force coefficients are computed from look-up tables or parametric expressions derived from geometric tools such as OpenVSP (NASA OpenVSP Team, 2025). The effect of actuator dynamics is captured using first- or second-order response models, governed by user-defined time constants and damping ratios. This introduces realistic response delays and rate limits to control surface inputs. Thrust is generated by propellers whose outputs are mapped from normalized throttle commands via

first-order response curves. In asymmetric thrust configurations, this can introduce differential yaw moments. Our simulator also supports ground effect modeling, which alters the lift and drag characteristics of the vehicle when flying close to the surface. This effect is modeled through empirical corrections (Phillips & Hunsaker, 2013) to the aerodynamic coefficients as a function of height-over-span ratio, tamper ration and aspect ratio. The detailed derivation and parameterizations for ground effect are provided in Appendix A. Overall, the simulator produces time-continuous dynamics that are discretized using a configurable integration scheme (e.g., Euler or RK4) and exposed through a modular interface supporting both CPU and GPU implementations. These dynamics form the basis for the environments used in training classical and learning-based controllers.

4 FALCON-S FRAMEWORK

Our simulation platform is designed to support the development, training, and evaluation of flight control strategies for fixed-wing aircraft operating in near-ground environments. The architecture, illustrated in Figure 1, consists of two primary modules: the *agent* and the *environment*.

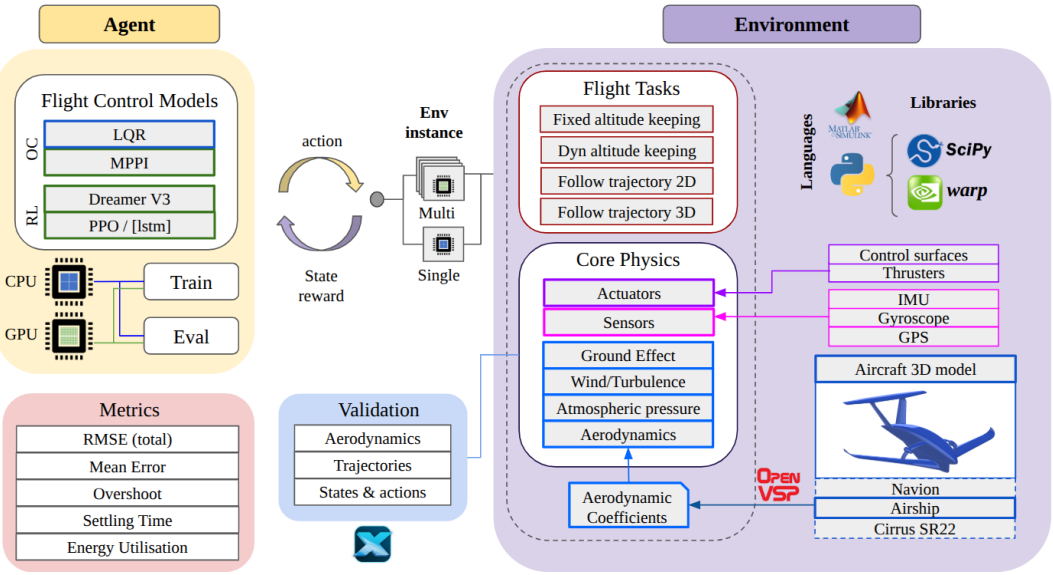


Figure 1: Overview of our FALCON-S simulation platform architecture. The environment module includes aerodynamic modeling, actuator dynamics, environmental effects such as ground effect and turbulence, and configurable sensor suites. The agent module supports both classical and learning-based controllers. Tasks, metrics, and visualization tools are modular and extensible, enabling robust benchmarking and policy training across single and multi-agent setups.

4.1 AGENT MODULE

The agent module supports a wide range of control models, including classical approaches such as Linear Quadratic Regulator (LQR) and Model Predictive Path Integral (MPPI), as well as modern learning-based controllers such as PPO, LSTM-based PPO, and DreamerV3. These controllers can be executed on either CPU or GPU for both evaluation and large-scale training, enabling both quick debugging of simulated flight conditions and heavy-duty batched experiments, where millions of trajectories can be collected to train and evaluate control performance.

LQR with Integral action: The LQR controller is implemented in closed form using discrete-time linearization of the vehicle dynamics around a steady trim condition. The gain matrix K is precomputed using the Riccati equation solution, and the resulting control law $u = -Kx$ is applied at each simulation step. The linearization matrices (A, B) are precomputed and approximated using numerical Jacobians based on the simulator's physics model. For LQRI (LQR with integral action) the state vector is augmented with integrator states (e.g. integrated altitude error) before forming (A_d, B_d) and solving the Ricatti equation B.1.

MPPI: The Model Predictive Path Integral controller follows a sampling-based trajectory optimization approach. At each control step, the algorithm samples multiple control sequences sampled from

a Gaussian distribution centred on the previous action sequence, propagates each action through the dynamics model, and computes an optimal control output from the weighted average of trajectories based on their cumulative cost. The implementation supports GPU-based sampling for parallelized inference, using a Warp backend. The cost function is task-specific and includes weighted penalties on tracking error, control effort, and constraint violations. More in Appendix B.2.

Gymnasium interface: The environment exposes a compliant Gymnasium Towers et al. (2024) interface through the `CoreAirshipEnv` class and its wrappers. It supports `reset()`, `step(action)`, and `render()` methods, and optionally includes `info` dictionaries with task-specific diagnostics. Observations are exposed as flat NumPy arrays and can be extended with sensor noise or delays via wrapper classes. The action space is continuous (bounded) and directly maps to control surface deflections and throttle values.

For high-performance training and evaluation, a parallelized variant of the environment is available through the Warp backend. This wrapper implements the same Gymnasium API but executes dynamics in batched form on the GPU, leveraging Warp's kernel-level integration and memory model.

Stable-Baselines3 and DreamerV3 support: We provide out-of-the-box integration with Stable Baselines3 (SB3), enabling rapid experimentation with off-the-shelf RL algorithms like PPO and SAC. Model-based RL agents are supported via a DreamerV3 (Hafner et al., 2025) pipeline that wraps the simulation environment in a recurrent state-space model (RSSM). The implementation reuses the 'dreamerv3' codebase, adapted for continuous-control fixed-wing tasks. The world model is trained jointly with a policy and value network using imagined rollouts. Action sequences are optimized through learned latent trajectories. GPU acceleration is used for both training and inference. See Appendix B.3 for architecture, hyperparameters, and adaptation details for fixed-wing control.

4.2 Environment module

Our environment module supports multiple simulation backends and interoperation with external tools, allowing flexibility in simulation fidelity, performance, and controller design workflows. Specifically, we offer two primary physics engines in Python: one based on SciPy's numerical integration for rapid prototyping, and another leveraging NVIDIA Warp for large-scale GPU-accelerated simulation. In addition, MATLAB and Simulink can be used for validation or control design tasks, such as symbolic derivation of system matrices for LQR or linearized model identification. This dual-language and dual-backend setup enables practitioners to prototype quickly in Python and validate or deploy controllers using industry-standard tools when necessary.

Core Physics: The environment simulates full six-degree-of-freedom (6DoF) rigid-body aircraft dynamics, focusing on low-altitude scenarios where physical effects such as ground proximity and turbulence dominate. The physics module is structured around five interconnected components:

Aerodynamics: Uses precomputed aerodynamic coefficients from OpenVSP or analytical approximations. Aerodynamic forces and moment forces are adjusted dynamically based on airspeed, angle of attack, sideslip angle, height above ground and control surface deflection. Ground effect corrections are applied using semi-empirical models A.1.

Actuators: Control surface deflections and thrust values are passed through first- or second-order actuator dynamics A.2, allowing simulation of latency, saturation, and rate-limited responses. The actuator module outputs net forces and moments in the body frame.

Environmental Effects: Wind gusts, turbulence fields, and pressure gradients are injected into the dynamics via different noise models A.3, enabling robustness testing under realistic conditions.

Sensors: An onboard sensor model simulates IMU measurements (accelerometer, gyroscope), GPS, and optional encoders A.4. Sensor noise, sampling rate, resolution, or delay can be added to evaluate performance under degraded sensing.

Flight Tasks: The agent interacts with the environment through a set of modular task definitions, such as fixed-altitude keeping, dynamic climbing/descending, 2D path following, and full 3D trajectory tracking. These are defined as reward functions and success conditions on top of the raw physics simulation.

Each of these components interacts through the environment interface, which passes state transitions, sampled observations, and reward signals to the agent. Each physics component can be independently toggled or simplified, enabling ablation studies and comparative benchmarking under controlled settings.

Aircraft 3D model: Our framework supports rapid prototyping of different airframes via JSON-based configuration files. Each aircraft model (e.g., Navion, Cessna, or Airship) is described by its

geometry, mass, inertia, control surface layout, propulsion system parameters, sensor configuration and environmental settings. Given the aircraft OpenVSP 3D model, using its python API, the aerodynamic coefficients can be computed as a luck-up table and then fitted to a Nth order polynomial. These models are then used for both simulation and visualization. The modular setup makes it easy to switch between vehicles and test control policies across different configurations, improving generalization and robustness.

Validation with X-Plane: To improve validation and high-fidelity visualization, FALCON-S includes an interface to the X-Plane using the Python XPlaneConnect API (NASA Ames Research Center, 2025) developed by NASA. Given the same aircraft configuration (geometry and flight initial conditions), trajectories generated in our simulator can be replayed or compared within X-Plane's high-resolution rendering engine. This allows cross-verification of dynamics between our model and an industry-standard closed-source simulator. Additionally, X-Plane can be used to test the different controllers and scenarios in an different simulation environment, providing a practical robustness and check for controller performance under a different modeling physics engine. Lastly, it can be used to capture high-quality video demonstrations of trained agents flying over varied terrain. In the Appendix A.5 we show examples for how comparisons with X-Plane can be performed.

5 Experiments & Results

Our experiments are designed to highlight the flexibility and realism of the FALCON-S framework, rather than to optimize or compare specific learning or control algorithms. The primary objective is to demonstrate how the simulator supports a wide variety of use cases and provides structured tools to evaluate control performance under diverse settings. To this end, we present a set of illustrative results covering four key aspects:

(1) **Algorithm performance illustration**: We demonstrate how FALCON-S supports consistent benchmarking by applying both classical (e.g., LQR) and learning-based (e.g., DreamerV3) controllers to standard tasks like altitude keeping.
(2) **Multi-task generalization**: We test a single controller (e.g., MPPI or LQR) on multiple tasks (e.g., altitude regulation, 2D path tracking, 3D trajectory tracking) to show how FALCON-S supports task variation and behavioral analysis with minimal reconfiguration.
(3) **Cross-vehicle testing**: Using the same control policy, we evaluate performance across different aircraft models (e.g., Cessna, Navion, Airship) to highlight how simulation fidelity and control difficulty change across morphologies and configurations.
(4) **Environmental sensitivity**: We analyze the impact of physical realism features, such as wind disturbance, ground effect, sensor noise, or actuator delay, by toggling them independently and observing the effect on controller robustness and behavior.

Metrics: To evaluate controller performance, we compute a set of standard metrics from each simulated trajectory, including root mean square error (RMSE), settling time, overshoot, energy utilization, and mean error. RMSE and mean error quantify overall tracking accuracy; settling time measures how quickly the agent enters and remains within a defined error band (1m); overshoot reflects the maximum deviation from the reference; and energy utilization serves as a proxy for control effort, computed from the squared motor actions over time. These metrics, together with full trajectory and action logs, allow structured comparisons across algorithms, tasks, vehicle models, and environmental settings. Equations are provided in the Appendix A.6.

Trajectories: The trajectories (a)–(f) correspond to: (a) altitude sine wave, (b) altitude ramps, (c) altitude and lateral ramps, (d) lateral sine wave, (e) altitude and lateral sine wave, and (f) spiral wave.

Each subsection below presents a brief experiment showcasing these capabilities. We leave detailed quantitative benchmarking and algorithm tuning to future work.

5.1 Demonstrating Learning-Based Control with Dreamer

To illustrate how FALCON-S supports modern reinforcement learning pipelines, we trained a DreamerV3 agent to perform altitude regulation. The task consists of maintaining flight along a forward trajectory while matching a time-varying altitude reference. Figure 2 shows the learned behavior over multiple rollouts, with 3D trajectory tracking, orientation stabilization, position evolution, and linear velocity regulation. The results indicate stable control behavior and successful

learning of the target altitude profile, albeit with slight oscillations due to limited policy tuning. Performance metrics across representative environments are summarized in Table 2, demonstrating tracking accuracy in the sub-meter range with energy usage that can be improved.

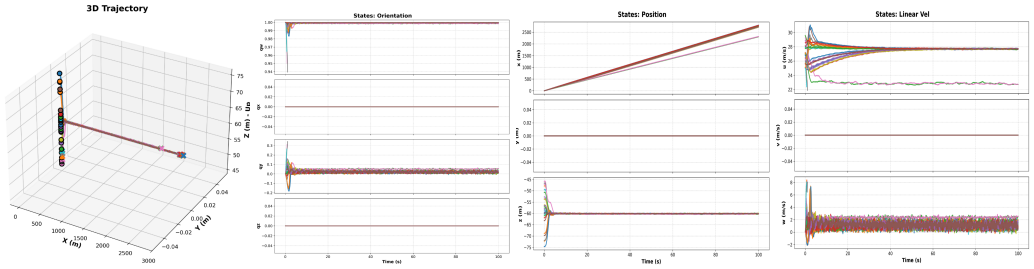


Figure 2: DreamerV3 agent controlling the airship along a dynamic altitude-keeping trajectory. Top-left: 3D trajectory tracking. Top-right: orientation convergence. Bottom-left: position over time. Bottom-right: body-frame velocity components.

Table 2: Mean ± standard deviation across five DreamerV3 runs for dynamic altitude tracking.

RMSE (m)	Alt. RMSE (m)	Overshoot (m)	Error (mean ± std) (m)	Energy Util.
0.756 ± 0.763	1.309 ± 1.231	1.540 ± 1.246	1.270 ± 1.369	0.743 ± 0.066

5.2 Single Controller Across Multiple Tasks

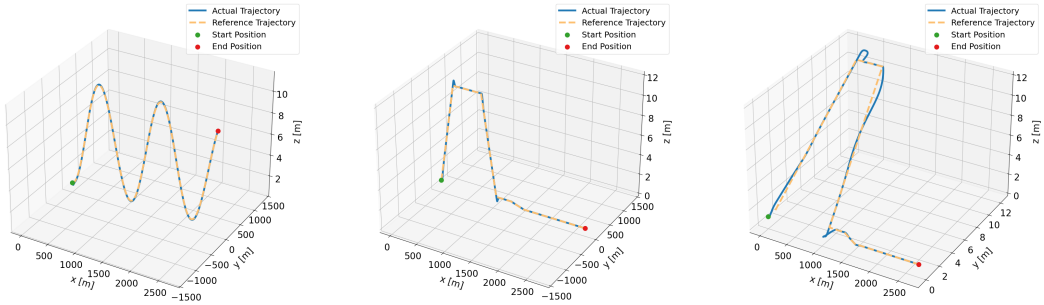
We evaluate the LQR controller on six trajectory tracking tasks of increasing complexity using the same airship model. As shown in Figure 3 and Table 3, the controller maintains low RMSE and smooth behavior on simpler tasks such as single-axis sine waves (a, d) and low-frequency ramps (b, e), with minimal overshoot and low energy usage. Performance degrades in more challenging 3D or fast-changing trajectories (c, f), where the controller exhibits larger errors and reduced stability. These results demonstrate the ability of our framework to highlight task-dependent control limitations and enable fine-grained benchmarking across diverse reference profiles.
Performance metrics for the MPPI controller can be seen in table 9 in section B.2 of the appendix.

Table 3: Performance metrics (RMSE, settling time, σ, overshoot, mean error, and energy utilization) for the Airship vehicle in tasks (a)–(f) with the LQR controller.

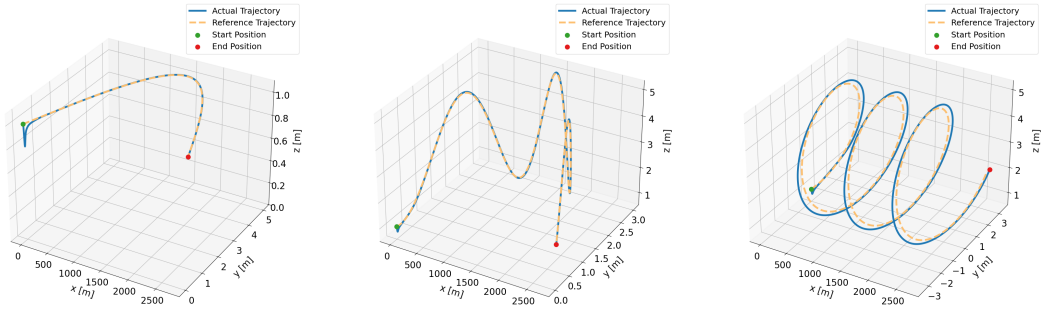
Task	**RMSE (m)**	**Settling Time (s)**	**Overshoot (m)**	**Error (mean ± std) (m)**	**Energy Utilization**
(a)	0.025	1.48	0.211	0.037 ± 0.024	0.663
(b)	0.052	41.55	0.671	0.025 ± 0.087	0.370
(c)	0.213	44.06	1.708	0.144 ± 0.339	0.379
(d)	0.017	2.94	0.213	0.014 ± 0.026	0.303
(e)	0.019	2.40	0.209	0.024 ± 0.021	0.583
(f)	0.149	–	0.704	0.231 ± 0.114	0.565

5.3 Cross-Aircraft Evaluation

To evaluate generalization across vehicle morphologies, we test the same LQR controller on three aircraft models, Airship (A), Cirrus SR22 (B), and Navion (C), across all six trajectory tracking tasks. As shown in Table 4, performance varies significantly with aircraft dynamics. The Airship (A), for which the controller was tuned, consistently achieves the lowest RMSE and overshoot, indicating good stability and responsiveness. In contrast, the Cirrus (B) and Navion (C) exhibit higher errors and settling times, especially in dynamic or multi-axis tasks (e.g., tasks c and f), due to differences in actuation and inertia properties. These results illustrate how the framework enables structured comparisons across vehicle configurations and supports benchmarking controller robustness to morphology changes.

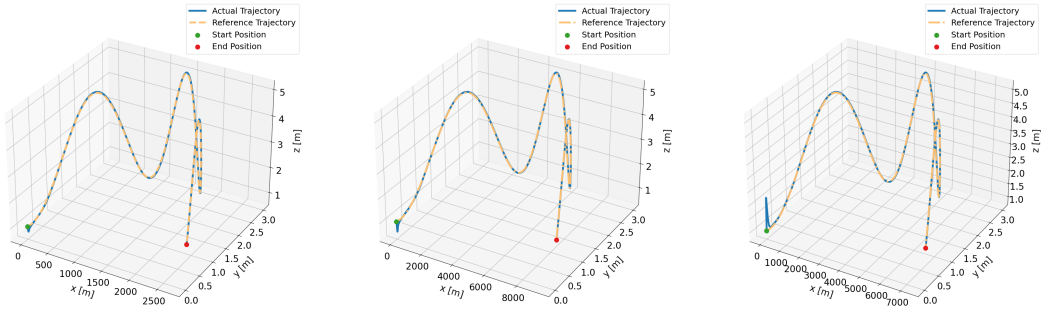


(a) Altitude sine wave (b) Altitude ramps (c) Altitude and lateral ramps

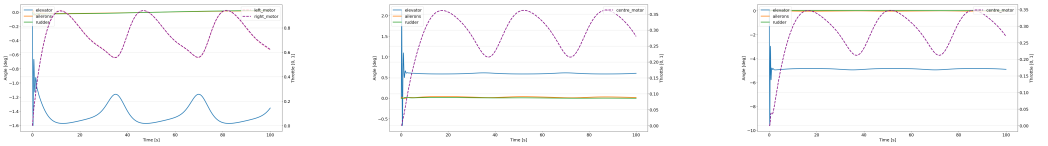


(d) Lateral sine wave (e) Altitude and lateral sine wave (f) Spiral wave

Figure 3: LQR-controlled Airship response to tracking different trajectories.



(a) Airship trajectory (b) Cirrus SR22 trajectory (c) Navion trajectory



(d) Airship actuator actions (e) Cirrus SR22 actuator actions (f) Navion actuator actions

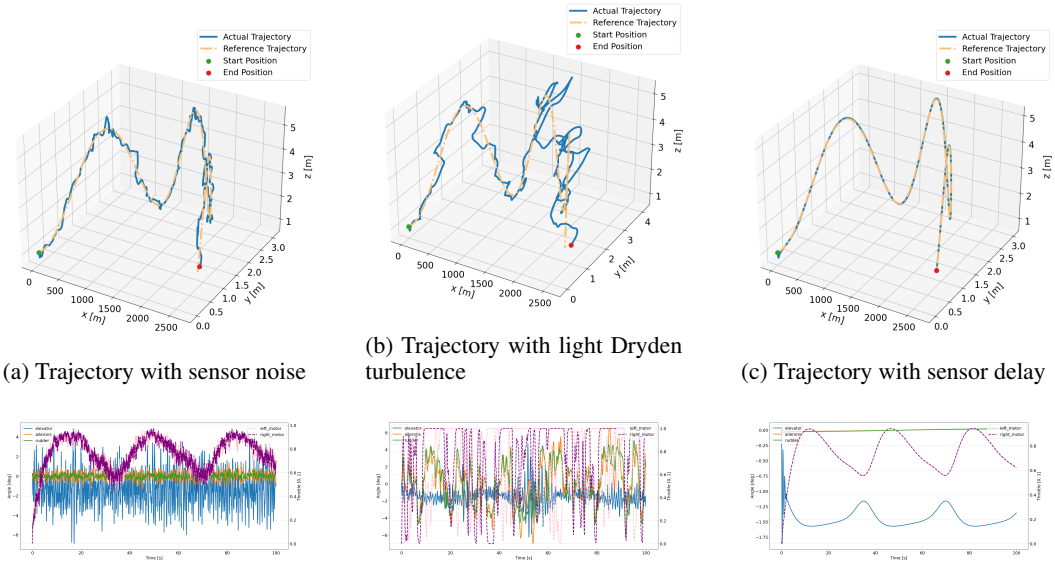
Figure 4: LQR-controlled Airship, Cirrus SR22 and Navion response to altitude and lateral sine wave trajectory tracking.

5.4 Robustness Under Environmental Variations

We assess the robustness of the LQR controller under different sources of environmental uncertainty: sensor noise (A), wind disturbances (B), and sensor delay (C). Table 5 shows that all three perturbations impact performance to varying degrees, with wind disturbances generally inducing the highest errors, overshoot, and energy usage, especially in fast-changing tasks such as (c) and (f). Sensor noise introduces more variability (e.g., increased RMSE and error variance), while sensor

Table 4: Performance metrics for scenarios (a)–(f). Columns (A), (B), (C) correspond to Airship, Cirrus SR22 and Navion respectively.

	RMSE			Settling Time (s)			Overshoot			Error (mean ± std) (m)			Energy Utilization		
Task	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)
(a)	0.025	0.024	0.055	1.48	1.63	1.51	0.211	0.419	1.227	0.037 ± 0.024	0.028 ± 0.031	0.034 ± 0.089	0.663	0.372	0.362
(b)	0.052	0.045	0.049	41.55	41.27	41.27	0.671	0.787	0.854	0.025 ± 0.087	0.019 ± 0.076	0.019 ± 0.082	0.370	0.322	0.315
(c)	0.213	0.223	0.208	44.06	46.20	46.00	1.708	1.709	1.877	0.144 ± 0.339	0.158 ± 0.352	0.148 ± 0.328	0.379	0.322	0.315
(d)	0.017	0.023	0.056	2.94	3.22	2.70	0.213	0.412	1.250	0.014 ± 0.026	0.017 ± 0.037	0.022 ± 0.094	0.303	0.290	0.289
(e)	0.019	0.022	0.055	2.40	2.59	1.95	0.209	0.416	1.237	0.024 ± 0.021	0.021 ± 0.032	0.027 ± 0.091	0.583	0.345	0.338
(f)	0.149	0.161	0.163	–	–	–	0.704	0.739	1.290	0.231 ± 0.114	0.248 ± 0.126	0.243 ± 0.143	0.565	0.340	0.335



(a) Trajectory with sensor noise

(b) Trajectory with light Dryden turbulence

(c) Trajectory with sensor delay

(d) Actuator actions with sensor noise

(e) Actuator actions with light Dryden turbulence

(f) Actuator actions with sensor delay

Figure 5: LQR-controlled Airship response to altitude and lateral sine wave trajectory tracking under sensor noise, light Dryden turbulence, or a 20 ms sensor delay.

delay has a relatively smaller effect in most scenarios, though certain tasks (e.g., (b), (f)) remain sensitive. These results demonstrate FALCON-S's capacity to simulate realistic disturbances and evaluate controller sensitivity in a structured and reproducible way.

Table 5: Performance metrics for scenarios (a)–(f). Columns (A), (B), (C) correspond to Airship, with sensor noise, wind disturbances and sensor delay respectively.

	RMSE			Settling Time (s)			Overshoot			Error (mean ± std) (m)			Energy Utilization		
Scenario	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)
(a)	0.091	0.220	0.024	64.43	–	0.01	0.619	1.155	0.215	0.131 ± 0.089	0.312 ± 0.218	0.036 ± 0.023	0.760	0.741	0.663
(b)	0.087	0.141	0.051	31.00	42.24	30.70	0.689	0.783	0.663	0.118 ± 0.093	0.178 ± 0.168	0.024 ± 0.084	0.464	0.476	0.371
(c)	0.394*	0.250	0.210	–*	43.37	43.20	1.732*	1.753	1.686	0.473 ± 0.492*	0.249 ± 0.354	0.142 ± 0.335	0.799*	0.474	0.380
(d)	0.066	0.076	0.017	0.01	0.01	0.01	0.356	0.382	0.217	0.099 ± 0.057	0.114 ± 0.065	0.014 ± 0.026	0.425	0.430	0.303
(e)	0.068	0.194	0.018	0.01	–	0.01	0.438	1.167	0.214	0.102 ± 0.060	0.253 ± 0.220	0.023 ± 0.021	0.669	0.668	0.583
(f)	0.163	0.238	0.147	2.34	87.28	2.48	0.766	1.401	0.691	0.260 ± 0.112	0.327 ± 0.249	0.228 ± 0.113	0.649	0.665	0.565

6 Conclusions

We introduced FALCON-S, a modular and high-fidelity simulation benchmark for fixed-wing aircraft operating in ground effect. By combining realistic 6DoF dynamics, configurable actuator and sensor models, and support for both classical and learning-based controllers, FALCON-S enables structured benchmarking across tasks, vehicle types, and environmental conditions. Its dual CPU-GPU backends and Gym-compatible API make it suitable for scalable training, analysis, and cross-validation. Future work will extend the framework with path planning algorithms, real-world hardware integration, and more solutions to support sim-to-real transfer.

REFERENCES

Calin Basescu, Tianshu Li, Nikolay Atanasov, and Vijay Kumar. Post-stall landing of a fixed-wing uav using learned aerodynamic models and nonlinear mpc. *IEEE Robotics and Automation Letters*, 8(3):1681–1688, 2023.

Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. *NeurIPS*, 32, 2019.

Jon Berndt. Jsbsim: An open source flight dynamics model in c++. In *AIAA modeling and simulation technologies conference and exhibit*, pp. 4923, 2004.

Eivind Bøhn, Erlend M Coates, Signe Moe, and Tor Ame Johansen. Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In *2019 international conference on unmanned aircraft systems (ICUAS)*, pp. 523–533. IEEE, 2019.

Elliot Chane-Sane, Pierre-Alexandre Leziart, Thomas Flayols, Olivier Stasse, Philippe Souères, and Nicolas Mansard. Cat: Constraints as terminations for legged locomotion reinforcement learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 13303–13310. IEEE, 2024.

Agostino De Marco, Paolo Maria D'Onza, and Sabato Manfredi. A deep reinforcement learning control approach for high-performance aircraft. *Nonlinear Dynamics*, 111(18):17037–17077, 2023.

HL Dryden and AM Kuethe. *Effect of turbulence in wind tunnel measurements*, volume 342. US Government Printing Office, 1930.

Matteo El-Hariry, Antoine Richard, Vivek Muralidharan, Matthieu Geist, and Miguel Olivares-Mendez. Drift: Deep reinforcement learning for intelligent floating platforms trajectories. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 14034–14041. IEEE, 2024.

Franz Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Rotors—a modular gazebo mav simulator framework. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 46–51, 2016.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640:647–653, 2025.

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

Laminar Research. X-Plane Flight Simulator. `https://www.x-plane.com/`, 2024. Accessed: 2025-09-12.

Jiayuan Liu, Jemin Hwangbo, Jongwoo Lee, et al. Impact of dynamics randomization on policy transfer for quadrotors. *IEEE Robotics and Automation Letters*, 6(3):5300–5307, 2021.

Viktor Makoviychuk, Lukasz Wawrzyniak, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

NASA Ames Research Center. X-plane connect (xpc). `https://github.com/nasa/XPlaneConnect`, 2025. Accessed: 2025-09-12.

NASA OpenVSP Team. Openvsp: Nasa's open vehicle sketch pad. `https://openvsp.org/`, 2025. Accessed: 2025-09-12.

Xingyou Pan, Yikang Cui, Yi Zhang, et al. Warpdrive: Extremely fast end-to-end deep multi-agent reinforcement learning on a gpu. *arXiv preprint arXiv:2108.13976*, 2021.

Warren F Phillips and Douglas F Hunsaker. Lifting-line predictions for induced drag and lift in ground effect. *Journal of Aircraft*, 50(4):1226–1233, 2013.

George Richards, Roberto Naldi, and Shankar Sharma. Neural networks for aerodynamic modeling of post-stall fixed-wing flight. In *AIAA Scitech 2021 Forum*, pp. 1–13. AIAA, 2021.

Christoph Richter and Ruben Calix. Qplane: A reinforcement learning toolkit for fixed-wing aircraft simulation. In *Proceedings of the ACM Multimedia Systems Conference (MMSys)*, pp. 334–337. ACM, 2021.

Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In *Conference on Robot Learning*, pp. 1147–1157. PMLR, 2021.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032*, 2024.

Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics*, 34(6):1603–1622, 2018. doi: 10.1109/TRO.2018.2865891.

Zifan Xue, Ashish Kumar, Karttikeya Jatavallabhula, Pulkit Agrawal, et al. Neuralplane: Efficiently parallelizable platform for fixed-wing aircraft control. In *NeurIPS Datasets and Benchmarks*, 2024.

A Physical Modeling Details

This appendix provides the mathematical details behind the simulation environment used in the main paper. The simulator integrates a high-fidelity 6DoF flight model with realistic actuator dynamics, wind turbulence (Dryden), and optional ground-effect modeling. These models are configured via a modular system that supports controlled ablation studies and toggling of physical phenomena.

A.1 Aerodynamic Coefficients and Ground Effect

The aerodynamic forces and moments are computed from lookup tables or polynomial fits, using the local flow conditions:

$$\mathbf{F}_{\text{aero}} = qS \begin{bmatrix} -C_D \\ C_Y \\ -C_L \end{bmatrix}, \quad \mathbf{M}_{\text{aero}} = qS \begin{bmatrix} bC_l \\ cC_m \\ bC_n \end{bmatrix}, \tag{3}$$

where $q = \frac{1}{2}\rho V_a^2$ is the dynamic pressure, S is the reference wing area, b and c are the wingspan and chord, and C_i are the aerodynamic coefficients dependent on angle of attack α, sideslip β, and control surfaces δ_a (ailerons), δ_e (elevator) and δ_r (rudder).

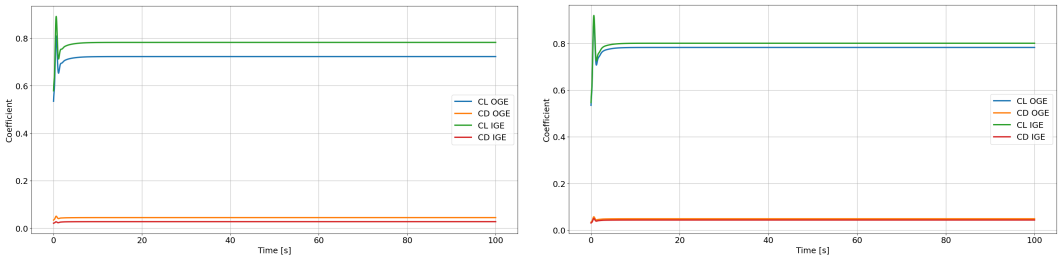
To model ground effect, the lift and drag coefficients C_L and C_D are corrected via empirical terms, following (Phillips & Hunsaker, 2013):

$$C_L = C_L^{\infty}\left(1 + \mu_L(h/b)\right), \tag{4}$$

$$C_D = C_D^{\infty}\left(1 - \mu_D(h/b)\right), \tag{5}$$

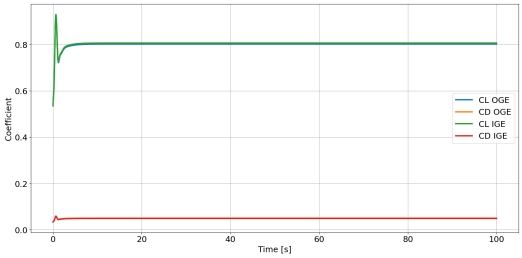
where μ_L, μ_D are ground effect modifiers parameterized as functions of the height ratio h/b, and $C_L^{\infty}, C_D^{\infty}$ denotes the out-of-ground-effect coefficients. These modifiers can be toggled to assess the effect of WIG-specific dynamics.

Table 6 presents the influence of ground effect on the performance of the LQR controller. As expected, operating close to the ground leads to a noticeable reduction in overall commanded thrust. The increase in C_L reduces the required angle of attack (α) for the same airspeed to maintain steady flight. Together with the higher μ_D in ground effect, this results in a substantial decrease in C_D, which lowers the overall drag and, consequently, the thrust required to sustain steady flight. These effect quickly become negligible once $(h/b) \geq 1$ (figure 6).

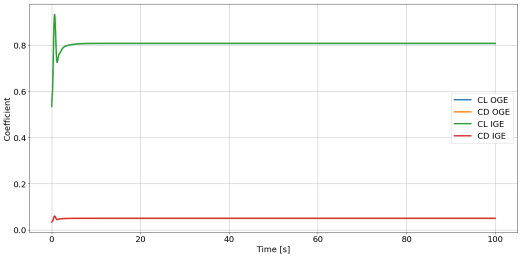


(a) 1.0 m constant altitude

(b) 2.5 m constant altitude



(c) 5.0 m constant altitude



(d) 10.0 m constant altitude

Figure 6: Variation of C_L and C_D with altitude due to ground effect, utilizing airship with the LQR controller.

Table 6: LQR performance metrics (RMSE, settling time, overshoot, error mean + std, and energy utilization) for the Airship vehicle at various altitudes.

Altitude (m)	RMSE (m)	Settling Time (s)	Overshoot (m)	Error (mean $\pm$ std) (m)	Energy Utilization
1.0	0.008	0.01	0.187	0.002 ± 0.014	0.303
2.5	0.011	0.01	0.247	0.002 ± 0.020	0.590
5.0	0.012	0.01	0.265	0.003 ± 0.021	0.689
10.0	0.013	0.01	0.271	0.003 ± 0.022	0.723
100.0	0.013	0.01	0.280	0.003 ± 0.022	0.743

A.2 Actuator Dynamics

Actuator systems (control surfaces and motors) are modeled via first- or second-order transfer functions with configurable time constants and damping ratios:

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \quad \text{(2nd-order)}, \tag{6}$$

or

$$H(s) = \frac{1}{\tau s + 1}, \quad \text{(1st-order)}, \tag{7}$$

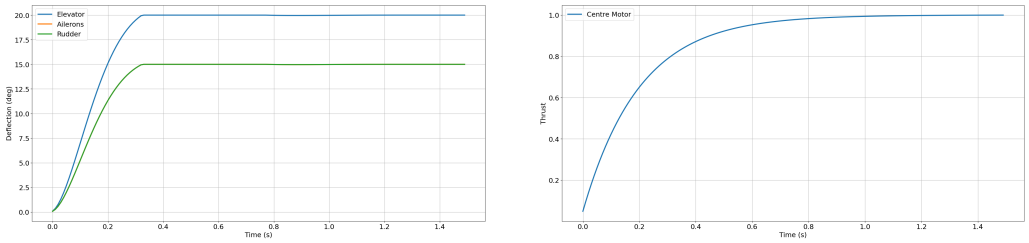
where ω_n is the natural frequency, ζ is the damping ratio, and τ is the time constant. Each actuator group (e.g., elevator, ailerons, motors) can use a different response model based on configuration.

A.3 Wind and Turbulence Modeling

Environmental disturbances include: - **Constant wind** in the inertial frame (NED), rotated to the body frame. - **Dryden turbulence** (Dryden & Kuethe, 1930), implemented via the MIL-F-8785C model using band-limited white noise through forming low-pass filters (see table 7).

For low altitude flights ($h < 1000\ ft$), the turbulence scale lengths and intensities are defined as

$$L_u = L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}, \quad L_w = h \tag{8}$$



(a) Control surfaces step response

(b) Motor step response

Figure 7: Control surfaces and motor unit step response for Airship.

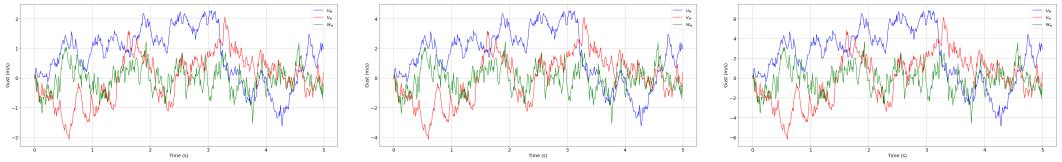
Table 7: Dryden turbulence velocity spectral filters.

	Longitudinal	**Lateral**	**Vertical**
Filter	$G_u(s) = \dfrac{\sigma_u K_u}{(1+T_u s)^2}$	$G_v(s) = \dfrac{\sigma_v K_v\left(1+\sqrt{3}\,T_v s\right)}{(1+T_v s)^2}$	$G_w(s) = \dfrac{\sigma_w K_w\left(1+\sqrt{3}\,T_w s\right)}{(1+T_w s)^2}$
Constants	$K_u = \sqrt{\dfrac{2L_u}{\pi\, U_0}},\quad T_u = \dfrac{L_u}{U_0}$	$K_v = \sqrt{\dfrac{L_v}{\pi\, U_0}},\quad T_v = \dfrac{L_v}{U_0}$	$K_w = \sqrt{\dfrac{L_w}{\pi\, U_0}},\quad T_w = \dfrac{L_w}{U_0}$

and

$$\sigma_u = \sigma_v = \frac{\sigma_w}{(0.177 + 0.000823h)^{0.4}}, \quad \sigma_w = 0.1W_{20}, \tag{9}$$

where h represents the altitude in feet, and W_{20} is the chosen wind speed at 20 meters, which defines the intensity of the turbulence.



(a) Light Dryden turbulence

(b) Moderate Dryden turbulence

(c) Severe Dryden turbulence

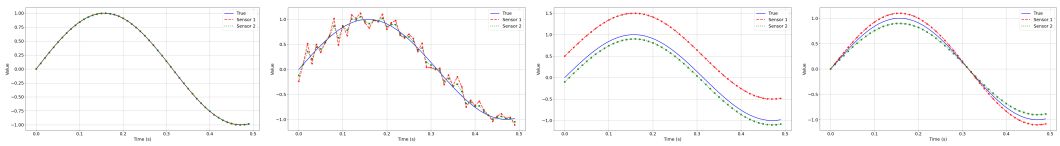
Figure 8: Longitudinal, lateral and vertical effects of different Dryden turbulence intensities at 2.5 m altitude, 28 m/s airspeed with the same random seed.

A.4 Sensor Realism and Noise

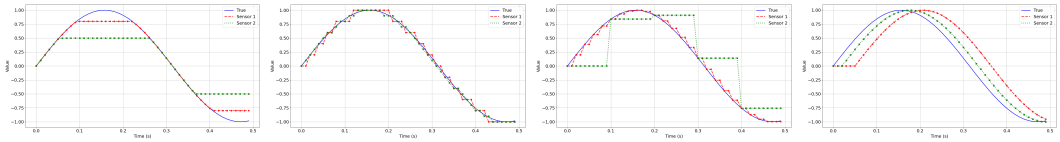
To simulate realistic perception pipelines, our framework includes configurable sensor models affected by various imperfections: additive noise, constant bias, scaling errors, clipping (limits), quantization (resolution), reduced sampling rates, and delay. Figure 9 illustrates these effects on sensor outputs compared to the ground truth signal. Each disturbance can be toggled or combined for robustness testing and sim-to-real transfer studies.

A.5 Validation with X-Plane

To support high-quality visualization and cross-simulator validation, FALCON-S includes a Python interface to X-Plane. This allows us to reproduce the same control task shown in previous experiments—such as dynamic altitude keeping—within X-Plane's rendering engine using the same aircraft configuration and reference trajectory. This enables visual inspection, qualitative validation, and future extensions toward sim-to-real transfer.



(a) Perfect sensors (b) Sensor noise (c) Sensor bias (d) Sensor scaling factor



(e) Sensor limits (f) Sensor resolution (g) Sensor sampling rate (h) Sensor delay

Figure 9: Illustration of different sensor effects implemented in the simulator.

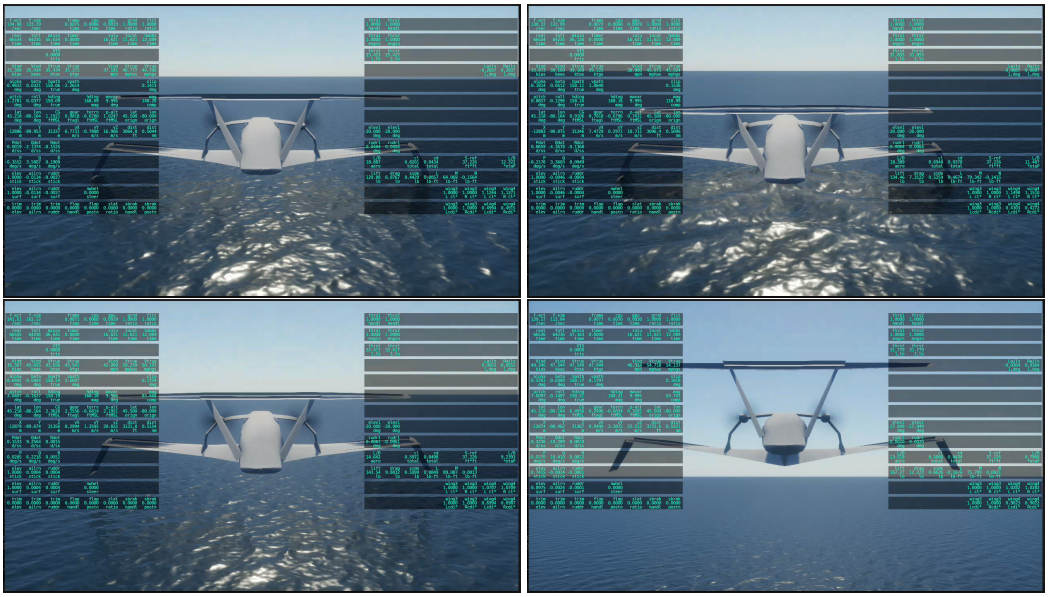


Figure 10: Rendered views of the Airship performing a dynamic altitude keeping task in X-Plane, aligned with the same reference trajectory used in FALCON-S.

A.6 METRICS

Table 8 summarizes the performance metrics used to evaluate control strategies in FALCON-S. Each metric is computed from logged trajectories and actions, capturing accuracy, responsiveness, and control efficiency.

For interpretation, lower values of tracking error and overshoot indicate higher accuracy, while shorter settling times reflect faster convergence. Control smoothness metrics (e.g., input rate penalties) capture responsiveness without excessive actuator usage. Energy consumption is evaluated from integrated thrust and control surface activity (range [0-1]): lower values indicate more efficient control. Conversely, excessively high energy consumption may reflect oscillatory or unstable control behavior.

Table 8: Summary of evaluation metrics computed from trajectory and control logs.

Metric	Formula / Description
RMSE (Total)	$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{t=1}^{N}\|\mathbf{e}_t\|^2}$ Where $\mathbf{e}_t$ is the position error at timestep t
Mean Error	$\text{Mean} = \frac{1}{N}\sum_{t=1}^{N}\|\mathbf{e}_t\|$
Overshoot	$\text{Overshoot} = \max_t \|\mathbf{e}_t\|$
Settling Time	Minimum time t such that $\|\mathbf{e}_t\| \leq \delta$ and remains within the band $\forall t' \geq t$ for at least 10% of the episode length. Default threshold: $\delta = 1\text{m}$
Energy Utilization	$\text{Energy} = \frac{1}{T}\int_0^T \|\mathbf{u}_{\text{motors}}(t)\|^2\, dt$ Where $\mathbf{u}_{\text{motors}}$ are normalized motor inputs and T is total time

B AGENT MODULE

B.1 LQR WITH INTEGRAL ACTION DETAILS

The Linear Quadratic Regulator (LQR) controller was tuned using state and input weighting matrices selected to balance tracking accuracy and control effort. The state weighting matrix Q_{LQR} was defined as

$$\begin{aligned} Q_{LQR} = \text{diag}(&14.6,\ 8.2,\ 14.6,\ 8.2,\ 14.6,\ 8.2,\ 1,\ 1,\\ &0.25,\ 0.25,\ 0.25,\ 18.2,\ 18.2,\ 18.2,\\ &10^{-5},\ 10^{-5},\ 10^{-5},\ 10^{-5},\ 1,\ 400), \end{aligned} \tag{10}$$

where the first six entries correspond to actuator states, followed by velocity, angular velocity, imaginary quaternion components, and position. The very small weights on the quaternion error terms $\left(10^{-5}\right)$ were introduced to avoid biasing the controller towards any one given attitude, while still ensuring stability.

To incorporate integral action, the augmented weighting matrix was defined as

$$Q_{\text{LQI}} = \text{diag}\ \left(Q_{\text{LQR}},\, Q_{\text{LQR}}(18{:}20)\right), \tag{11}$$

and the integral gains were set to $K_I = \begin{bmatrix} 0 & -1 & -1.5 \end{bmatrix}^\top$.

The control effort weighting matrix was chosen as

$$R_{\text{LQR}} = \text{diag}\big(1,\ 1,\ 800,\ 800,\ 800\big),$$

assigning higher penalties to thrust-related control inputs in order to limit excessive propulsive effort and improve efficiency.

The complete state-feedback gain matrix K (including integral augmentation where applicable) was computed in MATLAB using the built-in `lqr` function. The state-space matrices (A, B) required by `lqr` were obtained from the system linearization tool (linearization around the chosen trim condition). Full implementation and resulting K matrices for each aircraft can be seen in the open source repository.

B.2 MPPI DETAILS

For all experiments the MPPI controller was initialized the following settings: number of sampled trajectories $N = 10^3$, planning horizon $T = 100$ time steps, temperature $\lambda = 3.0$, and control perturbation covariance $\Sigma = \text{diag}(\sigma_e^2, \sigma_a^2, \sigma_r^2, \sigma_T^2) = \text{diag}(0.10,\ 0.08,\ 0.08,\ 0.10)$, for elevator, aileron, rudder and throttle respectively.

Table 9: Performance metrics (RMSE, settling time, σ, overshoot, mean error, and energy utilization) for the Airship vehicle for constant altitude and tasks (a)–(f) with the MPPI controller. Runs marked with '*' indicate simulations that terminated prematurely due to instability (crash or stall).

Task	RMSE (m)	Settling Time (s)	Overshoot (m)	Error (mean $\pm$ std) (m)	Energy Utilization
2.5 m altitude	0.010	0.84	0.119	0.013 ± 0.010	0.787
60.0 m altitude	0.013	1.08	0.158	0.017 ± 0.013	0.786
(a)	0.012	0.01	0.070	0.017 ± 0.011	0.780
(b)*	1.075	–	3.836	1.330 ± 1.303	0.367
(c)*	3.046	–	11.256	3.837 ± 3.621	0.340
(d)	0.008	0.01	0.081	0.011 ± 0.009	0.525
(e)	0.010	0.01	0.088	0.014 ± 0.010	0.757
(f)*	0.814	–	4.581	0.971 ± 1.023	0.419

The cost function employed in the planner is the sum of weighted penalties for: (i) altitude tracking, (ii) lateral (cross-track) tracking, (iii) ground-collision/proximity avoidance, (iv) angular-rate intensity, (v) airspeed keeping, (vi) penalization of excessive altitude, (vii) excessive angle-of-attack (α), and (viii) excessive sideslip (β).

MPPI trajectory sampling and propagation were implemented using NVIDIA Warp to JIT-compile the simulation kernels and execute large numbers of trajectories in parallel on the GPU.

The formulation and implementation follow the approach described in Williams et al. (2018). Exact numeric weights, cost functions and process are also available in the open-source repository.

B.3 DreamerV3 hyperparams

DreamerV3	
Discount Factor (γ)	0.997
Replay Buffer Size	$2,000,000$
Batch Size	16
Sequence Length	1024
Updates per Environment Step	32
Model Size:	
RSSM Hidden Size	384
RSSM Deterministic Units	3072
Discrete Latents per State	24
MLP Units	384
CNN Depth	24