AIRSHIP

Funded by the European Union's
Horizon Europe programme under
the grant agreement No 101096487

D5.3 Report on SW-in-the-loop and HW-in-the-loop testing

WP5 Control avionics

Disclaimer

The content of this deliverable reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

Document Information

Grant Agreement / Proposal ID	101096487
Project Title	AUTONOMOUS FLYING SHIPS FOR INTER-ISLAND AND INLAND WATERS TRANSPORT
Project Acronym	AIRSHIP
Project Coordinator	Universidad Politécnica de Madrid
Project starting date (duration)	1st January 2023 (48 months)
Related Work Package	WP5
Related Task(s)	T5.3
Lead Organisation	Universidad Politécnica de Madrid
Contributing Partner(s)	LU, TAU
Due Date	30/06/2026
Submission Date	30/06/2026
Dissemination Level	PU

Revision History

Revision	Revision Date	Name	Position
1.0	01/06/2026	Zorana Milosevic, Sergio Dominguez (UPM)	Researcher
2.0	15/06/2026	Pedro Lima, Matteo El Hariry (UL)	Researcher

TABLE OF CONTENTS

1 Introduction 7
1.1 Purpose of the Deliverable . . . 8
1.2 Document Structure . . . 8

2 Digital Twins 9

3 Software-in-the-loop 11
3.1 Introduction . . . 11
3.2 SIL architecture . . . 11
3.3 Results and conclusions . . . 13

4 Hardware-in-the-loop 15
4.1 Introduction . . . 15
4.2 HIL architecture . . . 15
4.2.1 HIL validation on the A0-S prototype . . . 17
4.3 Results and conclusions . . . 19

5 RL Control: Modelling, Training and Validation 21
5.1 Modelling and training environment . . . 21
5.2 Controllers . . . 21
5.3 Control results . . . 22
5.4 Software-in-the-loop cross-validation . . . 25
5.5 Hardware-in-the-loop on the ironbird . . . 27

6 Conclusions and Future Work 29

References 30

List of Tables

Table 1 Abbreviation and Acronyms . . . 5

List of Figures

Figure 1 Digital twin of the AIRSHIP A1 vehicle. Left: AIRSHIP prototype model developed in *Plane Maker*. Centre and right: AIRSHIP A1 operating in the X-Plane 12 simulation environment. 9
Figure 2 A1 model in Gazebo with the sensor stack implemented: LiDAR, RADAR, FPV camera, stereocamera pair, GPS and laser altimeter. . . . 12
Figure 3 Basic SIL architecture scheme for the dual hybrid simulation setup using X-Plane 12 and Gazebo. . . . 12
Figure 4 3D reconstruction of a vessel using the stereocameras in the A1 prototype modeled in Gazebo. . . . 13
Figure 5 A vessel in front of our A1 model in Gazebo (left) and the DBSCAN clustering from the LiDAR data displayed in Rviz (right). . . . 13
Figure 6 Top row: La Gomera map in X-Plane 12 (left) and in Gazebo (right). Bottom row: Port of Granadilla map in X-Plane 12 (left) and in Gazebo (right). . . . 14

Figure 7 Avoidance maneuver recorder from the FPV camera of the WIG model for a vessel coming from the right performed with our hybrid SIL simulation setup. . . . 14
Figure 8 Raspberry Pi executing the control algorithms while X-Plane 12 runs on a separate workstation. . . . 16
Figure 9 Hardware architecture used for the HIL validation of the AIRSHIP A0-S prototype in June, 2026. . . . 17
Figure 10 HIL validation setup employed on the AIRSHIP A0-S prototype. . . . 17
Figure 11 HIL validation campaign conducted on the AIRSHIP A0-S prototype at Lake Mänttä, Finland, in June 2026. The photographs show the experimental setup, the mobile ground station used for software deployment and monitoring, the live first-person-view (FPV) video received during flight, and flight tests performed. . . . 19
Figure 12 Resource consumption of the Raspberry Pi while executing the control algorithms in real time and communicating with the X-Plane 12 simulator running on a separate workstation. 20
Figure 13 Flight-software stack architecture. The control runtime (reference generation, controller, safety supervisor) is decoupled from the actuator transport, so the same stack drives either real PWM hardware or the simulation command topic feeding X-Plane through XPlaneConnect. . . . 22
Figure 14 LQR closed-loop tracking on representative manoeuvres: soft landing, steep up–down climb with coupled lateral motion, and a low-altitude (1 m) holding pattern. Actual (solid) versus reference (dashed) trajectories. . . . 23
Figure 15 LQR waypoint-tracking under Monte-Carlo sampling of random waypoint sets (heading-rate $< 5.7°$, altitude-rate < 0.5 m/s). The airship reaches the complete waypoint sequence (left, centre) with the corresponding attitude and actuator state (right). . . . 23
Figure 16 Learned altitude-keeping policy on the A0-S over randomly sampled altitude references: 3-D and side-view flight paths (colour encodes altitude), commanded-vs-achieved altitude traces, representative control inputs, and the Monte-Carlo tracking error. The settled error stays in the sub-metre range with no stall or crash. . . . 24
Figure 17 Altitude-keeping robustness sweep on the A0-S: convergence of the altitude error over many targets and randomised spawns (left), and the per-target settled RMSE distribution (right). Tracking is consistent across the whole altitude band. . . . 24
Figure 18 Closed-loop tracking error over time for the classical baseline controllers (LQR vs. MPPI) on the reference tracking task. Both settle to a low residual error. . . . 25
Figure 19 Tracking RMSE of the classical baseline controllers (LQR vs. MPPI) decomposed by axis. The two designs are within a few millimetres of each other, establishing the deterministic performance baseline against which the learned policy is compared. . . . 25
Figure 20 Software-in-the-loop with the unchanged control stack driving the AIRSHIP airframe in X-Plane 12 through XPlaneConnect. Left: constant low-altitude (1 m) tracking. Right: a ramp up-and-down trajectory, with the live pose estimates and control-surface/throttle commands logged during the run. . . . 26
Figure 21 Cross-validation of the in-house simulator against JSBSim under identical zero-input free-flight conditions. Blue: in-house simulator; orange: JSBSim. Altitude and pitch-rate correlations exceed 0.99 throughout. . . . 27
Figure 22 The ironbird bench: the AIRSHIP avionics (Cube autopilot, PWM driver, RC multiplexer, GPS/receiver and antennas, voltage regulators and servo BEC) mounted in flight configuration with the airframe absent, driven in real time by the simulator over the hardware-in-the-loop link. . . . 28

Abbreviations and Acronyms

Acronym	Description
AI	Artificial Intelligence
CPU	Central processing unit
DoF	Degrees-of-freedom
GNSS	Global Navigation Satellite System
GPU	Graphics processing unit
HIL	Hardware-In-the-Loop
IMU	Inertial Measurement Unit
LQR	Linear Quadratic Regulator
LQRI	Linear Quadratic Regulator with Integral action
MPPI	Model Predictive Path Integral
PPO	Proximal policy optimization
RL	Reinforcement Learning
SAC	Soft Actor Critic
SIL	Software-In-the-Loop
UWV	Unmanned WIG Vehicle
WIG	Wing-in-ground

Table 1: Abbreviation and Acronyms

Executive Summary

This deliverable reports the work carried out under Task 5.3 – WIG Software-in-the-Loop and Hardware-in-the-Loop Simulations (M12–M42), led by UPM and conducted in collaboration with UL, with support from TAU. It presents the research, development activities, and simulation results obtained for the different AIRSHIP system components using Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) methodologies.

Task 5.3 builds upon the work performed in the previous tasks of WP5. In particular, Task 5.1 – WIG System Modelling established the foundation by developing the initial virtual models of the vehicle, while Task 5.2 – WIG 6 DOF Control focused on the development and testing of control algorithms. During Task 5.2, the vehicle model was continuously refined and updated as new information became available from the evolving design and experimental testing of the AIRSHIP platform.

The initial dynamic model developed in Task 5.1 (M1–M12) provided a necessary baseline for simulation-based development. However, this early model was created when the AIRSHIP vehicle design was still at a conceptual stage. Consequently, it has been significantly refined and extended throughout Task 5.2, incorporating updated aerodynamic, structural, propulsion, and control system characteristics as the vehicle definition matured. These improvements were essential to enable realistic simulations, meaningful controller design, and reliable performance assessment.

Building on these developments, Task 5.3 established a progressive validation framework in which software components are first verified in simulation environments, then evaluated using increasingly realistic digital representations of the vehicle, and finally integrated with real hardware through Hardware-in-the-Loop testing. This approach reduces development risk, facilitates early detection of integration issues, and supports the verification of critical functionalities.

In addition to supporting the development and validation of low-level control systems, the simulation frameworks developed under Task 5.3 provide a common validation environment for higher-level autonomy functions, including situational awareness, guidance, navigation, trajectory replanning, and obstacle avoidance. As such, they constitute a core enabling technology connecting the activities of WP5, WP6, and WP7.

1 Introduction

The present deliverable focuses on the development, implementation, and validation of the Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) simulation frameworks within the AIRSHIP project, whose overarching goal is to develop an innovative, environmentally friendly Wing-in-Ground (WIG) vehicle capable of safe, efficient, and autonomous operation. Within this broader context, the work presented in this report addresses one of the key challenges associated with the development of autonomous systems: the ability to verify and validate complex software and hardware components in representative operational conditions before deployment on physical platforms.

The development of autonomous WIG vehicles presents unique challenges due to the complex aerodynamic interactions associated with ground-effect flight, the coupling between flight dynamics and environmental disturbances, and the need to integrate multiple subsystems including guidance, navigation, control, perception, and communications. Testing such systems exclusively through physical experimentation is both costly and risky, particularly during the early stages of development when vehicle designs, software architectures, and control strategies are still evolving. Consequently, simulation-based development and progressive verification methodologies play a central role in reducing technical risk and accelerating system maturity.

To address these challenges, this deliverable presents the simulation and validation infrastructure developed within AIRSHIP, encompassing a range of environments with increasing levels of fidelity. The work includes the development of virtual test environments, digital twins of the AIRSHIP vehicle, Software-in-the-Loop testing frameworks, and Hardware-in-the-Loop integration platforms. Together, these tools enable the systematic evaluation of algorithms and subsystems under realistic operational conditions while maintaining a safe and cost-effective development process.

Particular attention is devoted to the creation of simulation environments capable of accurately reproducing the vehicle dynamics, sensor behavior, actuator responses, and environmental disturbances relevant to WIG operations. These environments provide the foundation for validating control algorithms developed in Task 5.2, situational awareness capabilities developed in Task 6.4, and higher-level autonomous functions developed in WP7, including guidance, navigation, mission planning, trajectory execution, trajectory replanning, and obstacle avoidance.

Furthermore, the developed frameworks support the gradual transition from purely virtual simulations toward real hardware integration through HIL testing, providing a critical bridge between software development and field demonstrations. Beyond control validation, the simulation environments developed in Task 5.3 also provide the experimental infrastructure required for the development and assessment of situational awareness capabilities addressed in Task 6.4. By enabling the realistic simulation of surrounding vessels, obstacles, environmental conditions, sensor observations, and AIS information, these environments support the development and validation of WIG-centred situational awareness models and their associated information fusion and updating mechanisms.

By establishing a comprehensive SIL and HIL validation framework, this deliverable contributes to the AIRSHIP project's broader objective of demonstrating autonomous ground-effect flight through a progressive verification and validation strategy that combines simulation, hardware integration, and real-world experimentation.

1.1 Purpose of the Deliverable

The purpose of this deliverable is to report the work performed under Task 5.3 – WIG Software-in-the-Loop and Hardware-in-the-Loop Simulations. It presents the development of the simulation environments, digital twin models, validation methodologies, and test scenarios used to verify the different AIRSHIP subsystems throughout the project lifecycle.

The reported work builds directly upon the dynamic modelling activities performed in Task 5.1 and the control system development activities carried out in Task 5.2. The simulation environments described in this document provide the primary platform for the development, testing, and validation of control, guidance, and autonomy functions. In addition, the Hardware-in-the-Loop infrastructures developed within Task 5.3 support the progressive integration of real system components, enabling verification under realistic operating conditions while minimizing technical and operational risks.

The SIL and HIL frameworks presented in this deliverable also provide essential support for Task 6.4 – Situational Awareness and Task 7.1 – Guidance. The developed simulation environments enable the generation of realistic operational scenarios involving traffic, obstacles, environmental disturbances, and AIS-based information sources, thereby supporting the development and validation of WIG-centred situational awareness models and information fusion methodologies. In addition, these environments provide representative test conditions for higher-level autonomous capabilities, including waypoint tracking, trajectory following, mission-level navigation, trajectory replanning, and obstacle avoidance.

1.2 Document Structure

This document is organized as follows:

- Section 2 Describes the digital twin models and simulation environments developed for the AIRSHIP platform, including vehicle dynamics, environmental modelling, sensor simulation, and subsystem representations.
- Section 3 Details the Software-in-the-Loop framework, including software architecture, integration approach, simulation scenarios, and the results obtained during subsystem testing and validation.
- Section 4 Presents the Hardware-in-the-Loop infrastructure, test benches and the corresponding validation results.
- Section 5 presents the University of Luxembourg contribution: the architecture for developing, training and validating the reinforcement-learning control algorithms, together with the classical baseline, the control results, and their software- and hardware-in-the-loop cross-validation.
- Section 6 Concludes the deliverable, highlighting the main achievements, lessons learned, and outlining future work.

2 Digital Twins

The digital twin developed within the AIRSHIP project provides the virtual environment used for the development, integration, and validation of the vehicle's software and hardware components. It forms the basis of the Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) frameworks presented in this deliverable, enabling representative testing before deployment on the physical prototype.

The work presented in this section builds upon the modelling activities initiated in Task 5.1 - WIG System Modelling, where the initial dynamic model of the AIRSHIP vehicle was developed (see Deliverable D5.1). As the vehicle design matured throughout the project, the simulation model was continuously refined to incorporate updates to the vehicle geometry, mass properties, propulsion system, aerodynamic characteristics, onboard equipment, and control interfaces. These design changes are described in detail in the WP3 deliverables, particularly D3.2 and D3.3, and were continuously reflected in the digital twin to ensure that the simulation remained representative of the latest AIRSHIP prototype.

The AIRSHIP dynamics are simulated using the commercial flight simulator X-Plane 12 (Laminar Research, 2024), which computes aerodynamic forces using Blade Element Theory (BET). This approach enables realistic modelling of Wing-in-Ground vehicle aerodynamics, particularly during low-altitude flight where ground-effect phenomena have a significant influence on vehicle behaviour. Since none of the available models within X-Plane adequately represents a WIG vehicle, a custom AIRSHIP A1 model was developed in *Plane Maker* by Tampere University using the latest vehicle geometry, aerodynamic analyses, and experimental data obtained throughout the project. Communication between X-Plane and the AIRSHIP software architecture is established through the NASA *XPlaneConnect* plugin (NASA, 2017), enabling real-time bidirectional data exchange with the ROS 2 framework via the UDP protocol.



Figure 1: Digital twin of the AIRSHIP A1 vehicle. Left: AIRSHIP prototype model developed in *Plane Maker*. Centre and right: AIRSHIP A1 operating in the X-Plane 12 simulation environment.

While X-Plane provides a high-fidelity representation of the flight dynamics, it offers only limited capabilities for modelling robotic sensors and their interaction with autonomous software. To address this limitation, a complementary digital twin was developed in the Gazebo robotics simulator. Gazebo provides native ROS 2 integration and realistic models of the onboard sensor suite, including IMU, GNSS, cameras, LiDAR, and other perception sensors. It also supports configurable sensor noise, update rates, communication delays, coordinate frame transformations, and interaction with external software modules. These capabilities make Gazebo particularly suitable for the development and validation of perception, sensor fusion, navigation, and situational awareness algorithms.

A high-fidelity simulation environment was therefore constructed by coupling X-Plane 12 with Gazebo, combining realistic flight dynamics with advanced sensor simulation and middleware integration. The resulting digital twin enables the simultaneous validation of flight dynamics, control algorithms, perception systems, situational awareness functions, and higher-level autonomous behaviours within a unified

SIL and HIL framework. This integrated environment forms the foundation for the validation activities presented in the following sections of this deliverable.

3 Software-in-the-loop

3.1 Introduction

Following the development of the AIRSHIP digital twin presented in the previous section, the next step consisted of establishing a Software-in-the-Loop (SIL) framework capable of validating the different software components of the system under representative operating conditions. While the digital twin provides realistic models of the flight dynamics, sensors, and operating environment, the SIL framework integrates these models with the AIRSHIP software architecture, enabling closed-loop testing of autonomous functionalities before deployment on physical hardware.

Software-in-the-Loop (SIL) testing is a widely adopted verification methodology in the aerospace, automotive, and robotics industries. In a SIL configuration, both the system under test and its surrounding environment are simulated, allowing software components to be developed, integrated, and validated without requiring the availability of the physical platform. This approach reduces development costs and risks while enabling rapid iteration, repeatable experiments, and testing under conditions that would otherwise be difficult, expensive, or unsafe to reproduce experimentally.

Within the AIRSHIP project, the SIL framework provides the primary environment for the development and validation of flight control, guidance, navigation, perception, situational awareness, and mission management algorithms. It also serves as an intermediate step before Hardware-in-the-Loop (HIL) testing and subsequent field demonstrations.

3.2 SIL architecture

As explained in previous paragraphs, SIL simulation is a crucial step in the development of any autonomous system. The results from these simulations may give the user a better understanding of the system and how it could later interact with the environment in a future real-world deployment.

Nonetheless, in order to really trust the results from simulation, often performed under ideal (or close to ideal) conditions, the simulated environment must be closely aligned with real-world conditions. In many cases, modeling the simulation environment and its communication architecture to adapt it to the desired system is one of the most challenging parts of the process.

In our project, we have extensively used the X-Plane 12 flight simulator to test our control algorithms. This environment produces very reliable physics simulator which perfectly aligned with the complex aerodynamics of the WIG vehicle. Nonetheless, the system has limitations when it comes to simulating the environment. In addition, even though we can obtain most of the typical avionics data from X-Plane, it is impossible to simulate the external sensors that our WIG prototype uses to perceive the information of its surroundings (e.g.: LiDAR, cameras, RADAR, GPS, etc.).

With the aim of having the most realistic simulation setup possible for the WIG, we have implemented a hybrid dual-simulator architecture using *Gazebo* in addition to X-Plane to emulate a fully configurable maritime environment and our WIG prototypes with the complete perception stack implemented on them (see Fig. 2).

Both simulators work simultaneously, where X-Plane 12 carries out the flight and dynamics simulation, while Gazebo emulates the maritime scenario and the sensors information receiving data from that environment. Just as in the real system, the communication between the two simulators is fully imple-

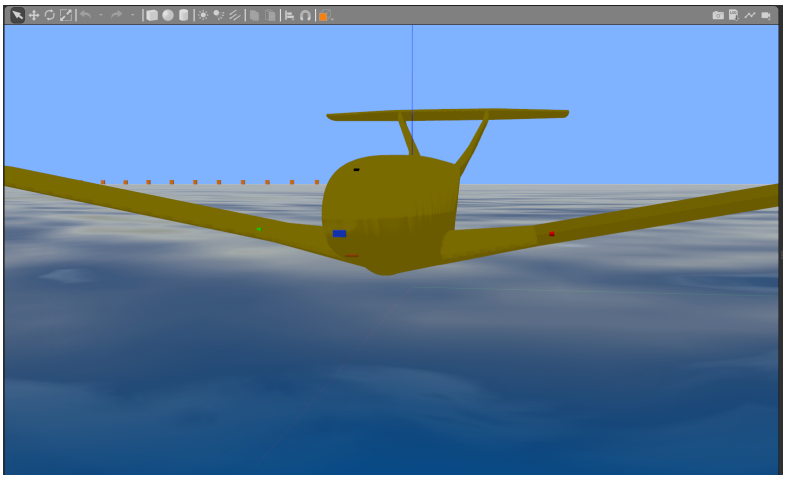


Figure 2: A1 model in Gazebo with the sensor stack implemented: LiDAR, RADAR, FPV camera, stereocamera pair, GPS and laser altimeter.

mented using the ROS 2 middleware, as well as the *NASA XPlaneConnect* plugin (NASA, 2017) to gather the data from the flight simulator (see Fig. 3).

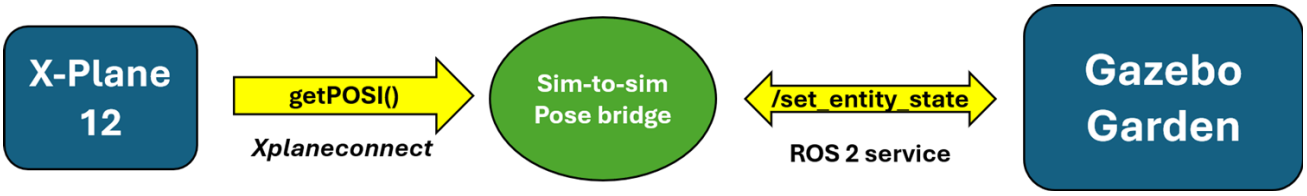


Figure 3: Basic SIL architecture scheme for the dual hybrid simulation setup using X-Plane 12 and Gazebo.

This simulation setup allows the user to test how the system interacts with its environment thanks to the constant flow of information coming from the modeled sensors mounted in the vehicle in Gazebo. As an example, in Fig 4, the emulated WIG vehicle performing Semi-global matching to reconstruct a vessel in front of it using the information from the stereo-cameras mounted on the wings of the vehicle.

Since we have also modeled the LiDAR and radar, we are also able to generate point clouds from the detections of these sensors in Gazebo, such as in Fig. 5, where a vessel is detected by the LiDAR and clustered using DBSCAN (Density-Based Spatial Clustering of Applications with Noise).

Using this hybrid simulation setup, we were also able to reproduce real-world scenarios by leveraging toolboxes developed by the broader Gazebo community, such as the Terrain Generator toolbox (Aravind, 2024). The integration and adaptation of this toolbox into our framework enabled the creation of a simulated environment that accurately reflects a real-world maritime setting. Given that one of the main focus of the project is the possible deployment of WIGs in EU's waterways with a high volume of freight transportation, we decided to recreate two ports from the Canary Islands, Spain (see Fig. 6).

Being able to test the capabilities of the WIG developed in this project in real-world scaled scenarios in simulation, makes the results coming from the SIL setup far more reliable since we are able to visually compute metrics such as distances required to execute maneuvers, distance to the water surface, real height of fixed obstacles (e.g.: rocks, mountains, etc).

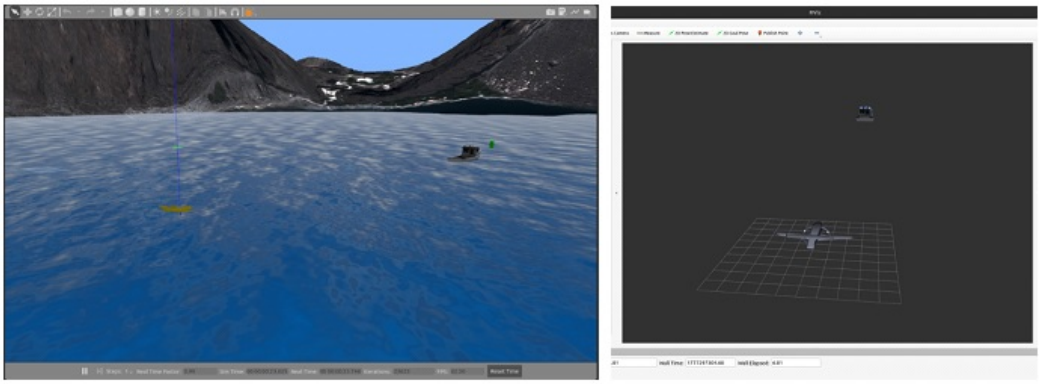


Figure 4: 3D reconstruction of a vessel using the stereocameras in the A1 prototype modeled in Gazebo.

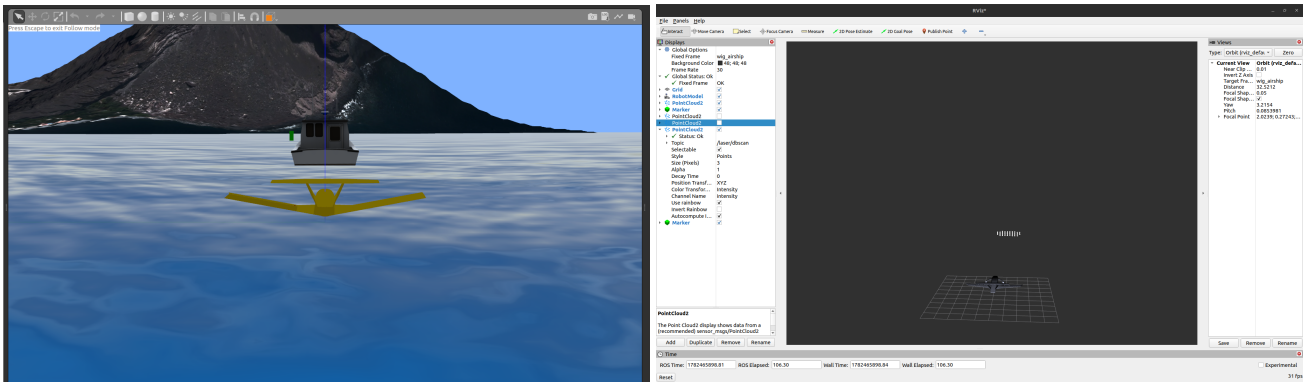


Figure 5: A vessel in front of our A1 model in Gazebo (left) and the DBSCAN clustering from the LiDAR data displayed in Rviz (right).

3.3 Results and conclusions

The proposed SIL setup has undoubtedly helped the progress of our research line by reproducing the behavior of not only the flight dynamics of the craft, but also its interactions with the surroundings.

The dual simulator architecture directly supports ongoing tasks such as *T6.4 Situational awareness*, where it is absolutely crucial to have data from the environment perceived by the sensors so that corrective actions can be applied to a moving WIG model, whose complex flight dynamics are carried out by X-Plane. Further work with this setup, including autonomous obstacle avoidance of fixed and dynamic obstacles, will be provided in deliverable D6.4 later this year. In Fig. 7, we present an avoidance maneuver performed by the WIG to avoid a collision with an incoming vessel.

Any test that requires reliable flight dynamics and interaction with real-world data can be perfectly performed in the presented setup, demonstrating the flexibility and capabilities of a solid SIL simulation environment in the robotics research field.

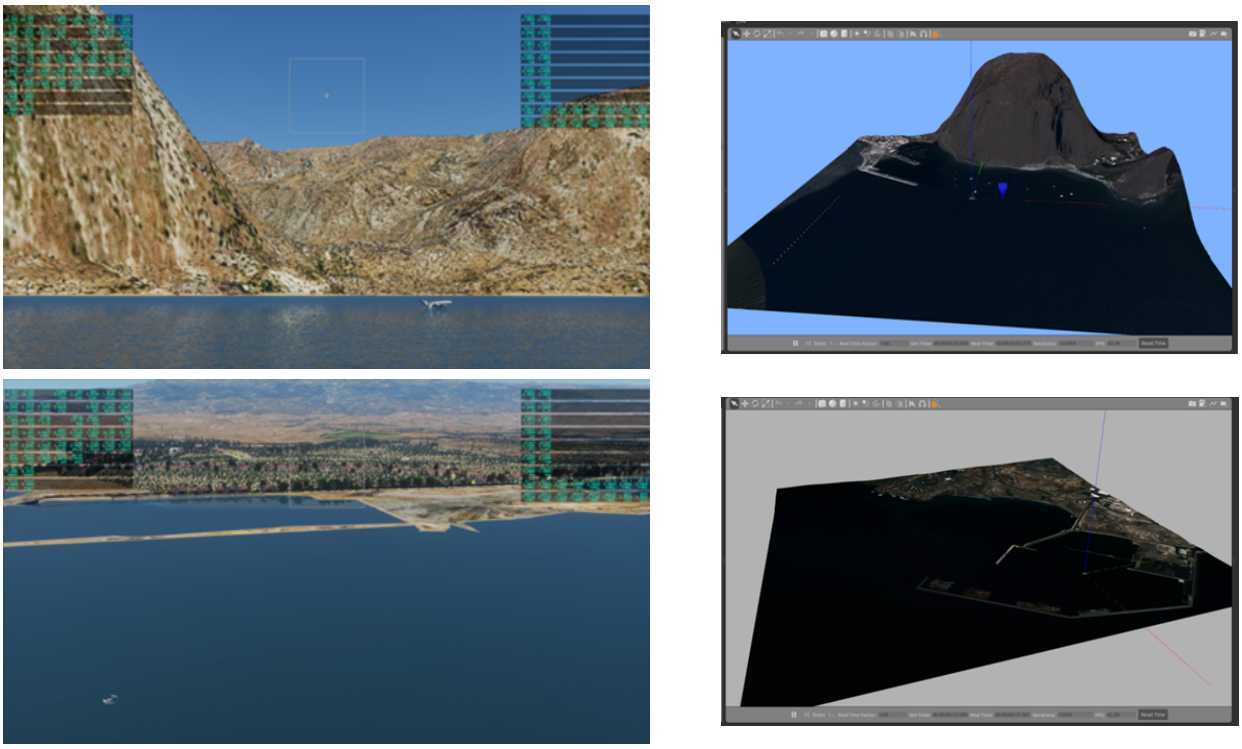


Figure 6: Top row: La Gomera map in X-Plane 12 (left) and in Gazebo (right). Bottom row: Port of Granadilla map in X-Plane 12 (left) and in Gazebo (right).

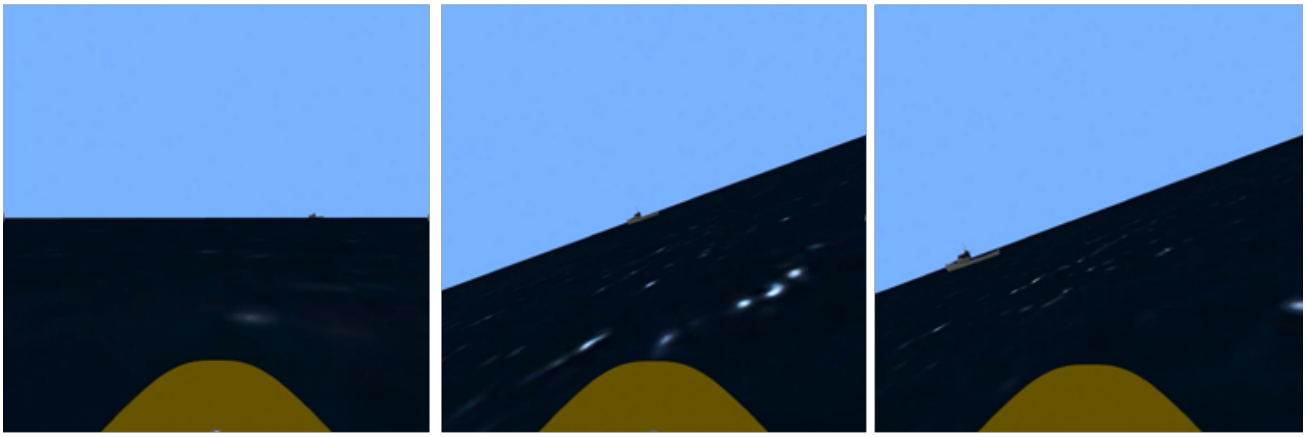


Figure 7: Avoidance maneuver recorder from the FPV camera of the WIG model for a vessel coming from the right performed with our hybrid SIL simulation setup.

4 Hardware-in-the-loop

4.1 Introduction

Following the development and validation of the SIL framework, the next step consisted of progressively replacing simulated components with real hardware, leading to the development of a Hardware-in-the-Loop (HIL) testing framework. While SIL enables the validation of software within a fully simulated environment, HIL allows critical hardware components to be integrated into the simulation while maintaining a safe and controlled testing environment.

HIL testing extends the concept of SIL by replacing selected simulated components with physical hardware devices. In a typical HIL configuration, the vehicle dynamics and operational environment remain simulated, while real hardware such as flight controllers, onboard computers, sensors, and communication interfaces interact with the simulation in real time. This approach enables the evaluation of hardware-software integration, communication latency, computational performance, and real-time execution before deployment on the physical vehicle.

Within the AIRSHIP project, the HIL framework was developed in parallel with the SIL environment to progressively validate the onboard computing architecture, flight controller integration, communication interfaces, and autonomous software stack. By introducing real hardware into the simulation loop, the framework provides a natural transition between purely virtual simulations and experimental flight testing while significantly reducing development risk.

4.2 HIL architecture

The HIL architecture was designed to closely resemble the software and hardware architecture of the AIRSHIP WIG prototypes while facilitating a progressive transition from the SIL framework to the final onboard system. By maintaining a common software architecture and communication framework throughout the development process, the transition from simulation to the physical platform can be achieved with minimal modifications to the software components.

The development of the HIL framework followed an incremental approach. Rather than integrating the complete hardware architecture from the outset, individual components were progressively introduced into the simulation loop. This allowed each subsystem to be validated independently before increasing the overall system complexity. The process began by introducing the onboard companion computer into the SIL framework, followed by the integration of the Pixhawk flight controller and, finally, the complete hardware architecture representative of the AIRSHIP prototype.

To facilitate this transition, the entire software architecture was developed using ROS 2 Humble as the common middleware, enabling fast and reliable communication between software modules, onboard computers, and flight hardware. Using the same communication framework for both SIL and HIL environments minimizes integration effort and allows software components developed in simulation to be transferred directly to the physical platform.

As an initial step, the onboard companion computer was integrated into the simulation loop while maintaining the flight dynamics inside X-Plane 12. Since this stage focused primarily on validating the software architecture and computational performance rather than the final hardware configuration, a Raspberry Pi 4B was selected as the companion computer due to its widespread availability, ease of integration, and native compatibility with both ROS 2 Humble and the Pixhawk flight controller.

Preliminary laboratory experiments were conducted to validate the software architecture and communication framework. In this configuration, the Raspberry Pi executed the control loops while the X-Plane 12 simulator ran on a separate workstation with greater computational resources. The overall communication architecture is shown in Fig. 8. The Raspberry Pi executed the control algorithms at 20 Hz and transmitted actuator commands to X-Plane via UDP over an Ethernet connection. After each simulation step, the updated vehicle state was transmitted back to the Raspberry Pi, where the next control action was computed. This configuration allowed the computational capabilities of the Raspberry Pi to be evaluated under real-time operating conditions while maintaining a high-fidelity simulation environment. The successful execution of these experiments demonstrated that the Raspberry Pi provides sufficient computational performance to support subsequent HIL validation activities.

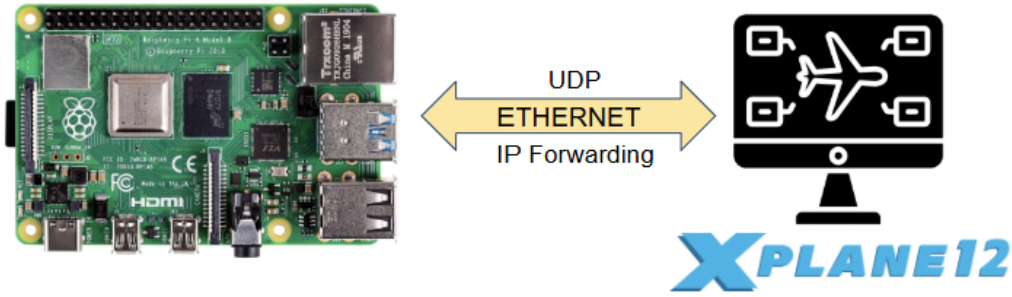


Figure 8: Raspberry Pi executing the control algorithms while X-Plane 12 runs on a separate workstation.

The next stage of development consisted of integrating the Pixhawk flight controller into the HIL framework. To communicate with the Pixhawk from the ROS 2 ecosystem, the *ArduPilot* firmware provides the *MAVROS* package (Ermakov & Community, 2024), which establishes a bridge between the native MAVLink protocol and ROS 2 topics. This enables ROS 2 nodes to exchange information with the flight controller using standard ROS communication mechanisms.

Communication between the Raspberry Pi and the Pixhawk was established through an Ethernet connection to minimize communication latency and ensure reliable real-time operation. This interface provides access to both external sensors connected to the Pixhawk, such as the GNSS receiver, airspeed sensor, and altimeter, as well as the internal sensors, including the IMU, barometer, and temperature sensor. Furthermore, the MAVLink–ROS bridge is fully bidirectional, allowing the Raspberry Pi not only to acquire sensor measurements but also to transmit control commands directly to the flight controller within the ROS 2 framework.

The final HIL architecture was designed to closely replicate the software and hardware configuration of the AIRSHIP A0-S prototype, as it formed the basis of the latest HIL validation campaign conducted in June 2026. As shown in Fig. 9, the proposed setup incorporates the principal onboard components, including the Pixhawk flight controller, Raspberry Pi companion computer, and the external flight sensors used during the experiments. Communication between the companion computer and the flight controller is established through a bidirectional MAVLink–ROS 2 bridge over Ethernet, allowing real-time acquisition of sensor data and transmission of control commands.

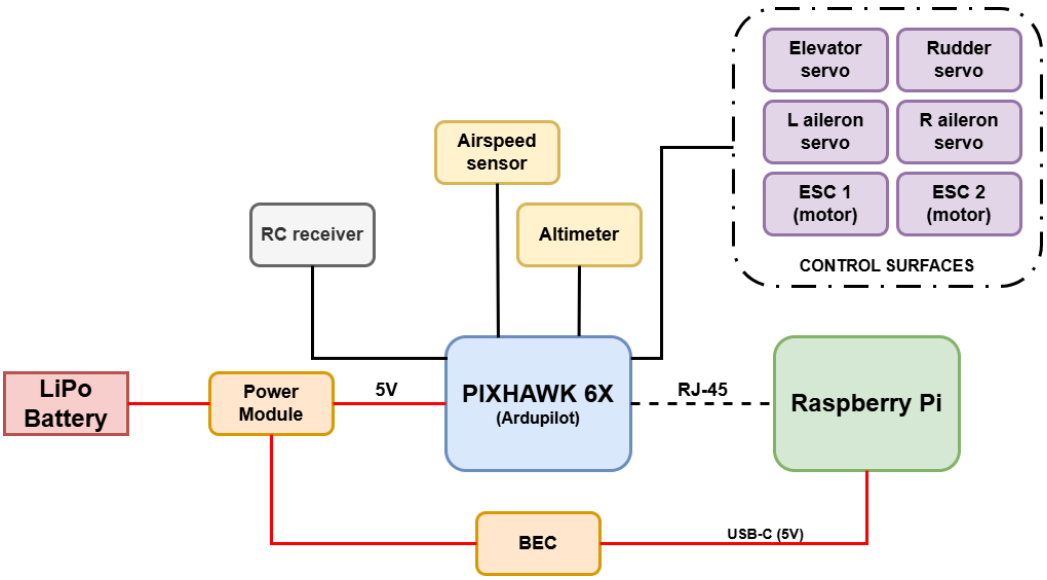


Figure 9: Hardware architecture used for the HIL validation of the AIRSHIP A0-S prototype in June, 2026.

4.2.1 HIL validation on the A0-S prototype

One of the key steps in transferring control and autonomy algorithms from simulation to a real platform is their validation on a scaled prototype before deployment on the final vehicle. This reduces development risk while allowing software and hardware integration issues to be identified and corrected under representative flight conditions. Within the AIRSHIP project, these HIL validation activities were carried out using the A0-S prototype.

The HIL architecture presented in the previous section was deployed on the A0-S prototype during the validation campaign. As illustrated in Fig. 10, two laptops were used to communicate with the onboard companion computer (Raspberry Pi 4), which was connected via Ethernet to the Pixhawk 6X flight controller. This configuration established a real-time ROS 2 communication network, allowing the onboard software to access flight data, execute the guidance, navigation, and control algorithms, and exchange information with the flight controller while preserving the pilot's ability to take manual control whenever necessary.

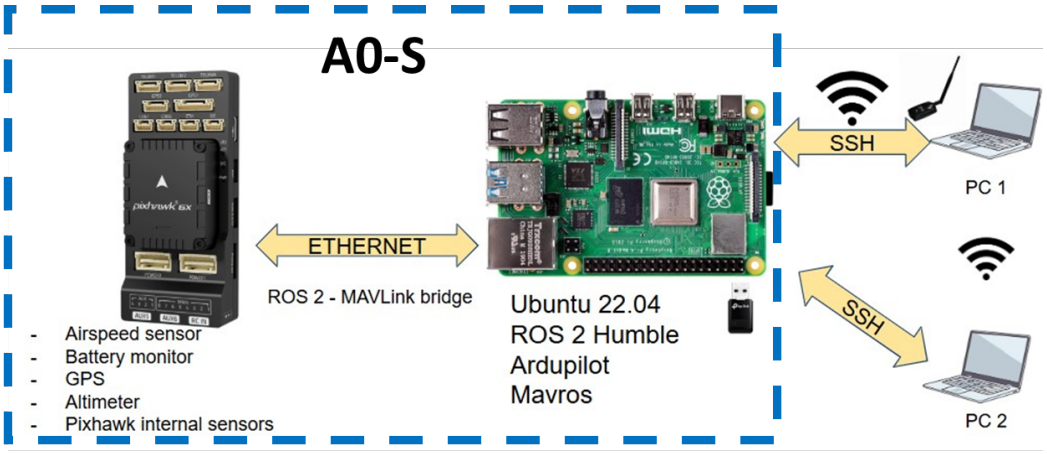


Figure 10: HIL validation setup employed on the AIRSHIP A0-S prototype.

To facilitate remote operation during the experiments, one of the laptops was configured as a Wi-Fi hotspot, allowing all devices except the Pixhawk to communicate over a common local network. This enabled remote access to the Raspberry Pi through the SSH protocol for software deployment, controller execution, and live monitoring of the system during flight.

The validation was conducted in *shadow mode*, where the autonomous guidance, navigation, and control algorithms executed in real time on the Raspberry Pi using live flight data while a human pilot retained full authority over the vehicle. The control commands generated by the autonomous system were logged but were not transmitted to the actuators. This approach enabled safe verification of software integration, state estimation, guidance logic, communication performance, computational timing, and control command generation under real flight conditions without introducing the risks associated with autonomous actuation.

Although shadow-mode testing does not allow the assessment of closed-loop control performance, stability, or trajectory tracking, it provides a safe intermediate validation step between laboratory HIL experiments and fully autonomous flight testing. Furthermore, it enables direct comparison between the pilot's control actions and the commands generated by the autonomous controller, providing valuable information for controller tuning, performance evaluation, and future autonomous flight experiments.



(a) Pre-flight preparation of the A0-S prototype.



(b) Mobile ground station used for software deployment and system monitoring.



(c) Live FPV video stream received during flight.



(d) A0-S prototype during the HIL validation flight.



(e) Representative flight of the A0-S prototype.

Figure 11: HIL validation campaign conducted on the AIRSHIP A0-S prototype at Lake Mänttä, Finland, in June 2026. The photographs show the experimental setup, the mobile ground station used for software deployment and monitoring, the live first-person-view (FPV) video received during flight, and flight tests performed.

4.3 Results and conclusions

The HIL validation activities demonstrated the feasibility of progressively transferring the AIRSHIP software stack from a fully simulated environment to representative flight hardware. By incrementally introducing the onboard companion computer, the Pixhawk flight controller, and finally the complete A0-S hardware architecture into the validation process, potential software integration and communication

issues were identified and resolved before experimental flight testing.

The preliminary laboratory experiments confirmed that the Raspberry Pi 4B provides sufficient computational performance to execute the guidance, navigation, and control algorithms in real time while maintaining stable communication with the X-Plane 12 simulator (Fig. 12). The integration of the Pixhawk through the MAVLink–ROS 2 bridge enabled the acquisition of real sensor measurements and their direct processing by the onboard software, providing a representative environment for validating the communication architecture and tuning the control algorithms before field deployment.

The subsequent HIL experiments on the A0-S prototype further demonstrated the correct integration of the onboard computing architecture with the flight controller under real operating conditions. Shadow-mode testing verified the complete software pipeline, including state estimation, communication, guidance, control command generation, and onboard execution, while maintaining safe manual operation of the vehicle. These experiments provided confidence that the software architecture can operate reliably on the target hardware and significantly reduced the technical risks associated with subsequent autonomous flight testing. Some representative photographs of the test campaign carried out with the AIRSHIP A0-S prototype at Lake Mänttä, Finland, in June 2026, are shown in Figure 11.

Overall, the HIL framework established in Task 5.3 constitutes an important intermediate step between SIL simulations and full-scale flight demonstrations. The developed infrastructure supports the validation of control, guidance, perception, and autonomy functions, and provides the foundation for the experimental demonstrations planned in Task 8.4.

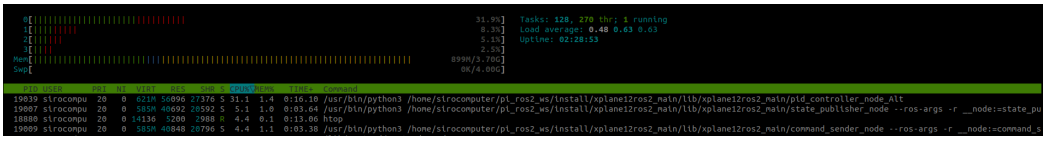


Figure 12: Resource consumption of the Raspberry Pi while executing the control algorithms in real time and communicating with the X-Plane 12 simulator running on a separate workstation.

5 Reinforcement-Learning Control: Modelling, Training and Validation

The control activities contributed by the University of Luxembourg (UL) complement the SIL and HIL frameworks of Sections 3 and 4 with a dedicated environment for developing, training and validating the low-level flight controllers of the AIRSHIP WIG vehicle. Both a classical baseline and learning-based policies are developed on the *same* 6-DoF dynamics and then promoted along the verification ladder: cross-checked in software against independent flight-dynamics engines, and finally exercised on the physical avionics. This section summarises the modelling environment, the controllers, their results, and the SIL/HIL steps specific to this contribution.

5.1 Modelling and training environment

All controllers were developed on a Gymnasium-compatible (Towers et al., 2024) 6-DoF vehicle environment that propagates the full rigid-body dynamics of the vehicle, including aerodynamics, atmosphere, force/moment build-up, quaternion attitude, ground effect and Dryden turbulence. The same physics definition backs two interchangeable back-ends: a GPU-accelerated kernel set used for large-scale reinforcement-learning training, and a CPU reference implementation. The two were checked to produce step-for-step identical states, so that a given result reflects a property of the vehicle dynamics rather than of a particular simulator kernel.

The controllers run inside a modular flight-software stack whose architecture is shown in Fig. 13. A state-source adapter fuses the incoming ROS2 (or X-Plane bridge) messages into a normalised vehicle state; a control runtime then orchestrates the periodic step that links reference generation, the selected low-level controller, and a safety supervisor enforcing arming, stale-state protection, clipping and command slew-rate limits. Only the resulting safe command is allowed to leave the core runtime, after which an actuator-output stage maps it either to real PWM hardware (through the PCA9685 channel mapping) or to a simulation command topic. This decoupling between the controller runtime and the actuator transport is what allows the *identical* control stack to be run in SIL, in HIL and on the vehicle without modification.

5.2 Controllers

Classical baseline (LQR). The deterministic baseline is a Linear Quadratic Regulator with integral action (LQRI) for altitude and heading hold. The nonlinear dynamics are linearised about the cruise trim to a linear time-invariant model $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}$, and the feedback gain minimises the usual quadratic cost on state and input (Kwakernaak & Sivan, 1972). The integral augmentation removes steady-state error under constant disturbances, and the controller is gain-scheduled on airspeed to cover the cruise envelope. LQR is the deployment baseline: it is cheap, interpretable, and maps directly onto the structure already present in standard autopilots such as ArduPilot and PX4.

Learning-based control (RL). The learning-based controllers are Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) policies (Raffin et al., 2021; Bøhn, Coates, Moe, & Johansen, 2019), trained on the batched GPU environment. The agent observes the vehicle state (altitude and airspeed errors, attitude, body rates) and outputs continuous control-surface and throttle commands; the reward favours accurate tracking while penalising stall and excessive control effort. Training is massively parallel, but the resulting policy is a small multilayer perceptron that, at inference, reduces to a single forward pass

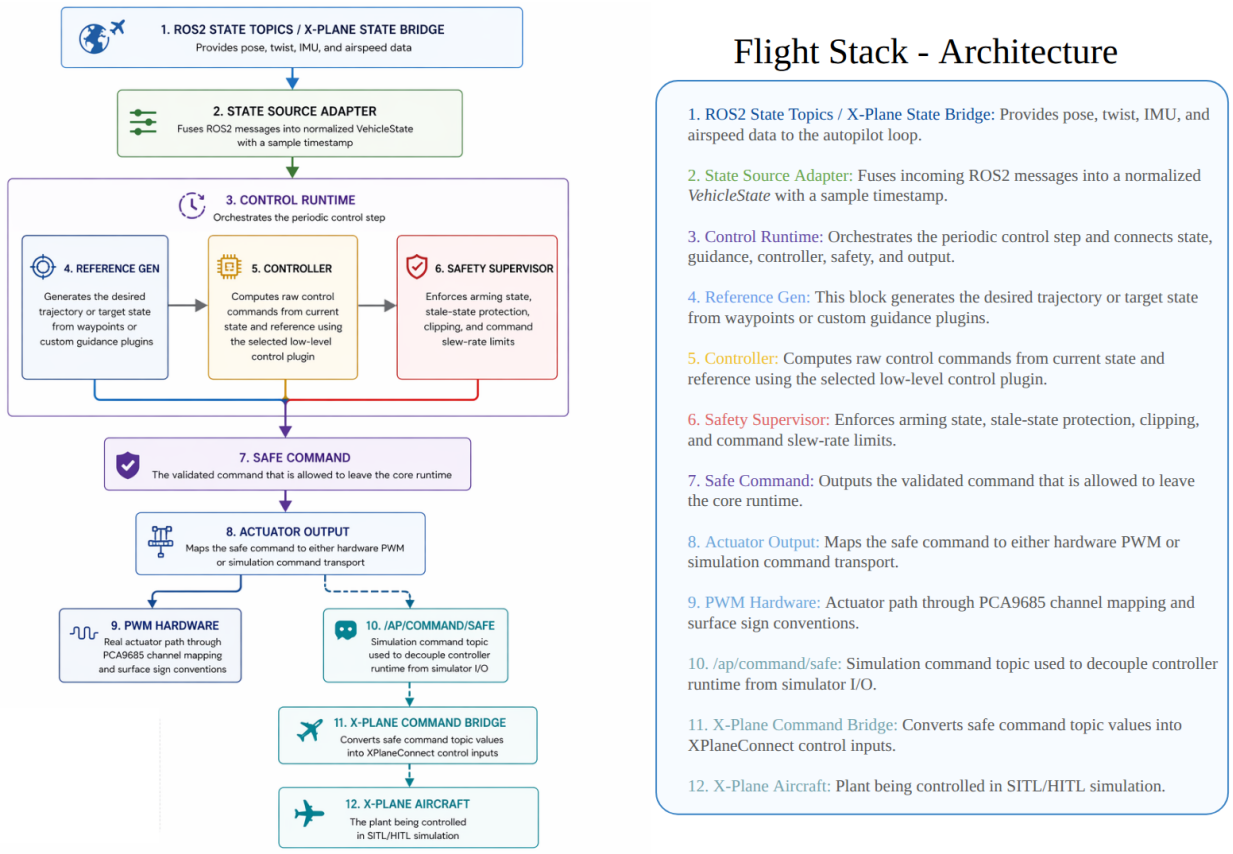


Figure 13: Flight-software stack architecture. The control runtime (reference generation, controller, safety supervisor) is decoupled from the actuator transport, so the same stack drives either real PWM hardware or the simulation command topic feeding X-Plane through XPlaneConnect.

light enough to run on a Pixhawk-class flight computer with no GPU and no deep-learning runtime. This keeps the learned controller compatible with the same onboard hardware used in the HIL framework of Section 4.

5.3 Control results

The classical baseline was first evaluated on a set of structured tracking tasks on the airship model. Figure 14 shows representative closed-loop manoeuvres — a soft landing, a steep coupled climb with simultaneous lateral motion, and a low-altitude (1 m) holding pattern — in which the controller follows the reference trajectory closely across both gentle and aggressive profiles. The controller was then exercised on randomly generated waypoint missions under Monte-Carlo sampling (Fig. 15), with heading-rate and altitude-rate constraints applied to the generated references; the airship reaches the full waypoint set in the reported cases, confirming the baseline is sound before the learned policy is introduced.

The learned altitude-keeping policy was then evaluated on the A0-S airframe over a Monte-Carlo set of altitude references (Fig. 16). The policy holds the forward flight path while matching a time-varying altitude command, with the settled tracking error in the sub-metre range and no stall or crash across the sampled targets, confirming that a single learned policy generalises across the cruise altitude band

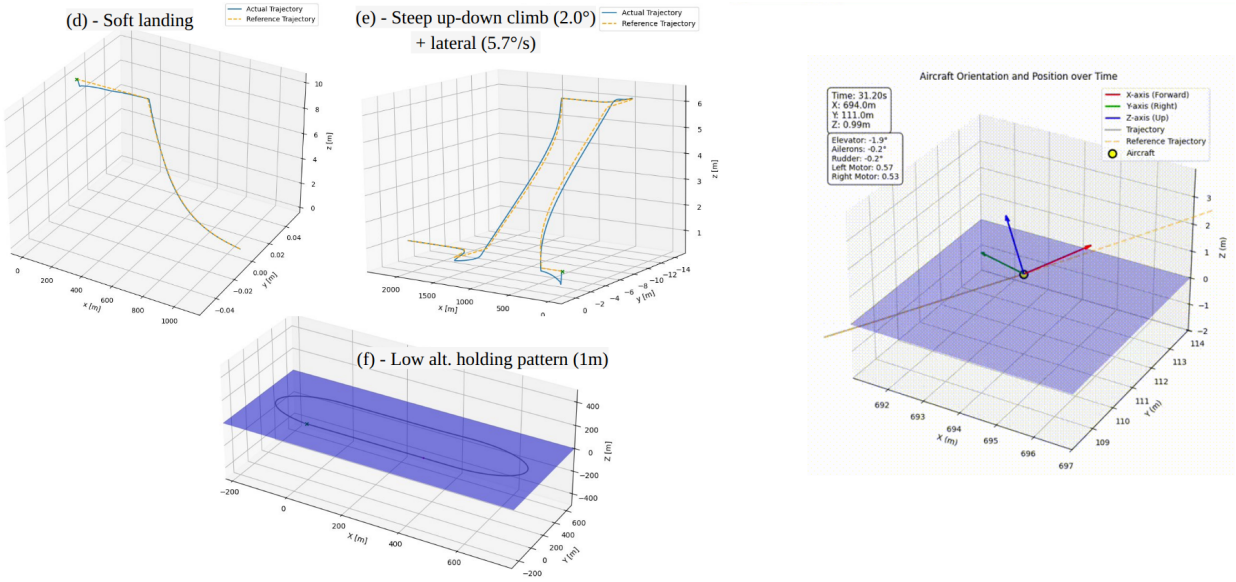


Figure 14: LQR closed-loop tracking on representative manoeuvres: soft landing, steep up–down climb with coupled lateral motion, and a low-altitude (1 m) holding pattern. Actual (solid) versus reference (dashed) trajectories.

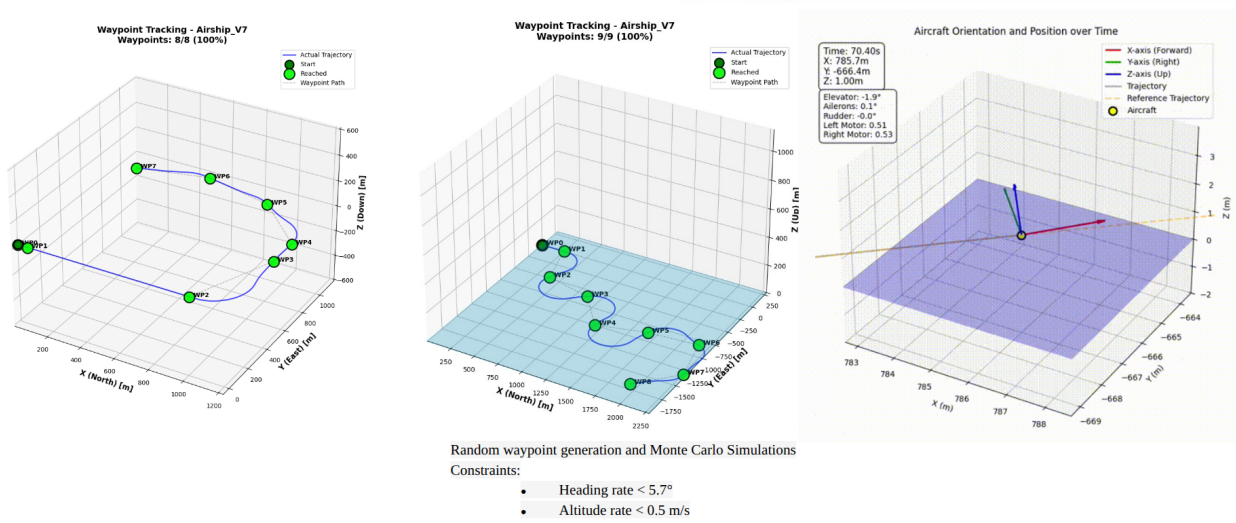


Figure 15: LQR waypoint-tracking under Monte-Carlo sampling of random waypoint sets (heading-rate $< 5.7°$, altitude-rate < 0.5 m/s). The airship reaches the complete waypoint sequence (left, centre) with the corresponding attitude and actuator state (right).

rather than being tuned to one set-point.

To confirm robustness rather than over-fitting, the policy was re-evaluated over a sweep of altitude targets and randomised initial conditions (Fig. 17). Convergence is consistent across the full envelope and the per-target settled RMSE remains small and roughly uniform.

Finally, the deterministic controllers were benchmarked head-to-head on the same tracking task. Figure 18 shows the closed-loop tracking error over time and Figure 19 the resulting RMSE for the LQR and the sampling-based Model Predictive Path Integral (MPPI) baselines. Both achieve low steady-state error; the learned policy matches this accuracy when stable while remaining robust to sensor noise

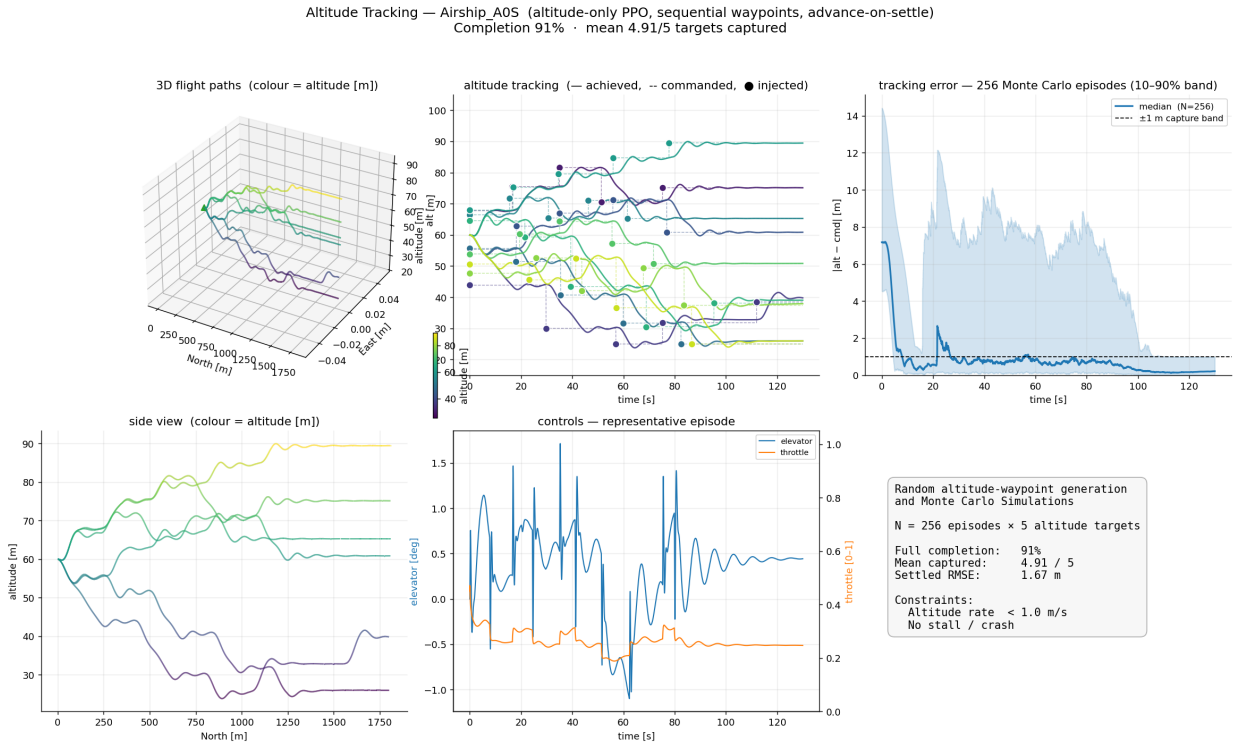


Figure 16: Learned altitude-keeping policy on the A0-S over randomly sampled altitude references: 3-D and side-view flight paths (colour encodes altitude), commanded-vs-achieved altitude traces, representative control inputs, and the Monte-Carlo tracking error. The settled error stays in the sub-metre range with no stall or crash.

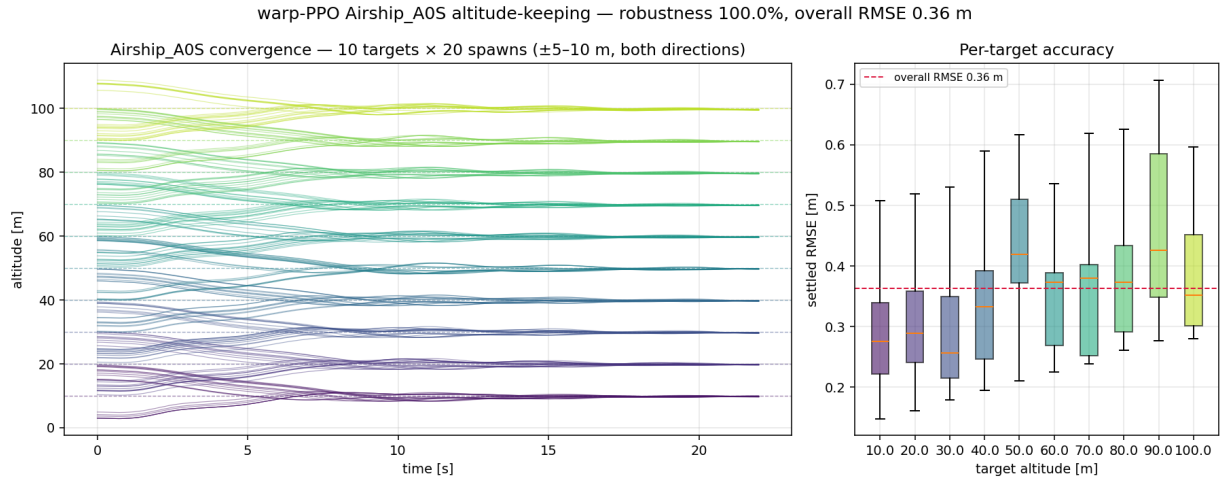


Figure 17: Altitude-keeping robustness sweep on the A0-S: convergence of the altitude error over many targets and randomised spawns (left), and the per-target settled RMSE distribution (right). Tracking is consistent across the whole altitude band.

and actuator lag, whereas the LQR rejects atmospheric turbulence where the nominal learned policy is more fragile. Neither design dominates, which motivates pairing a stabilising classical inner loop with the learned controller rather than relying on either alone.

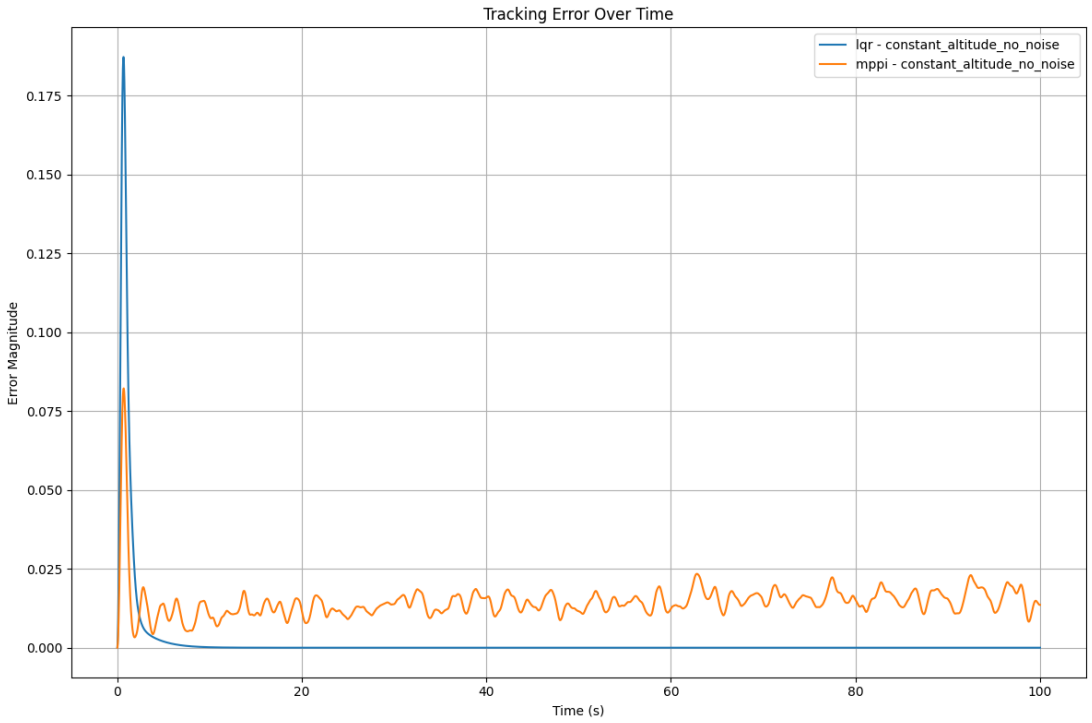


Figure 18: Closed-loop tracking error over time for the classical baseline controllers (LQR vs. MPPI) on the reference tracking task. Both settle to a low residual error.

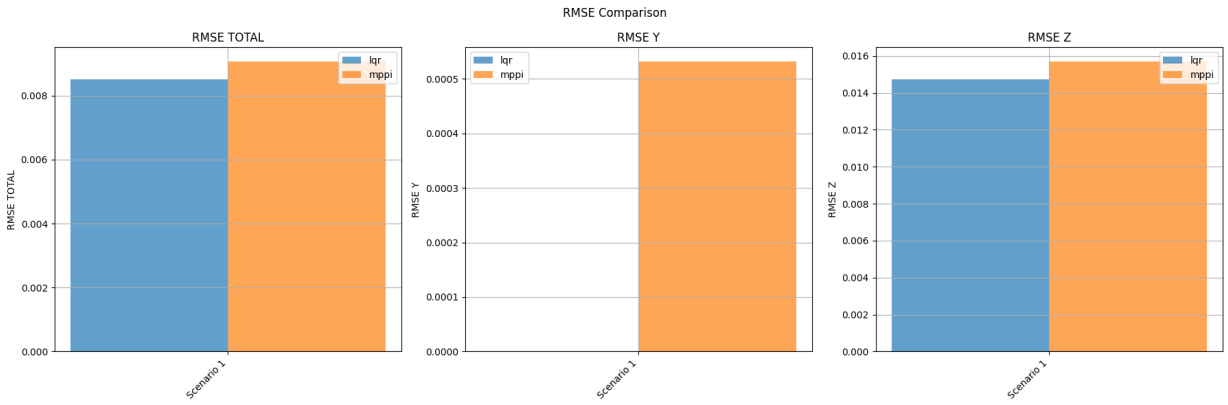


Figure 19: Tracking RMSE of the classical baseline controllers (LQR vs. MPPI) decomposed by axis. The two designs are within a few millimetres of each other, establishing the deterministic performance baseline against which the learned policy is compared.

5.4 Software-in-the-loop cross-validation

A control result obtained on the in-house simulator is only trusted once it is reproduced on an independent physics model. In addition to the X-Plane 12 coupling described in Section 3, the unchanged control stack is closed around the X-Plane 12 flight model directly through XPlaneConnect (Laminar Research, 2024; NASA, 2017): the same controller flies reference manoeuvres inside X-Plane, with the live pose estimates and control-surface commands logged for comparison against the in-house run (Fig. 20).

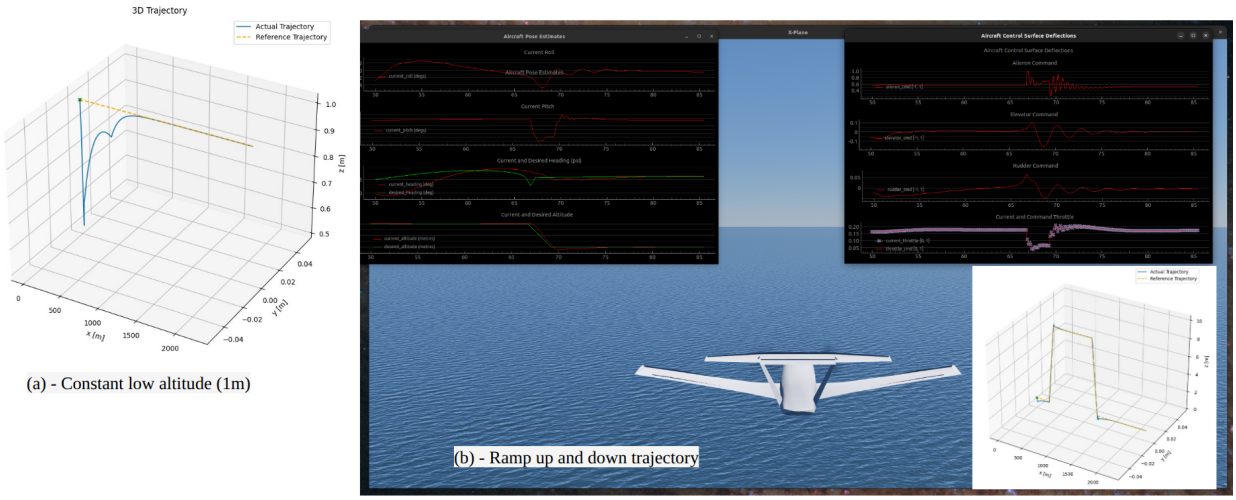
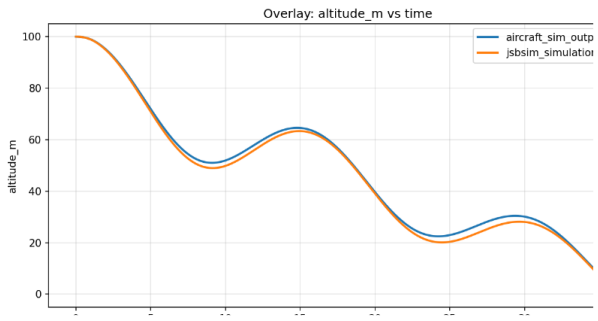
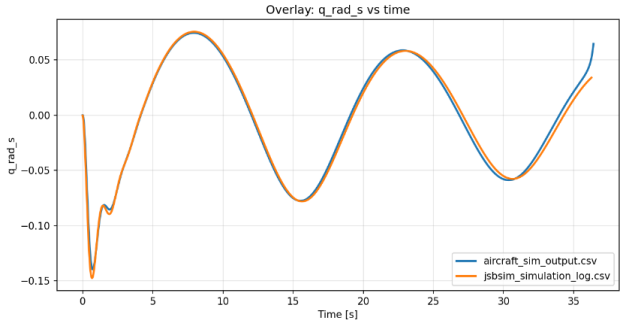


Figure 20: Software-in-the-loop with the unchanged control stack driving the AIRSHIP airframe in X-Plane 12 through XPlaneConnect. Left: constant low-altitude (1 m) tracking. Right: a ramp up-and-down trajectory, with the live pose estimates and control-surface/throttle commands logged during the run.

The control stack is additionally closed around *JSBSim* (Berndt & the JSBSim Development Team, 2011), an open-source 6-DoF flight-dynamics model, through its programmatic interface, giving a second independent cross-check. Under identical zero-input free-flight conditions the in-house and JSBSim trajectories agree closely: altitude and pitch-rate histories overlay across the episode with correlations above 0.99 (Fig. 21). With ground effect disabled the two models are in near-perfect agreement, isolating the in-house ground-effect correction as a genuine aerodynamic contribution rather than a numerical artefact. Using two engines of different lineage — one commercial and visual, one open and scriptable — provides stronger evidence than either alone: a trajectory reproduced on the AIRSHIP simulator, on X-Plane and on JSBSim is robust to the choice of physics model.



(a) - 100m initial altitude, no actuation - altitude plot



(b) - 100m initial altitude, no actuation - angular vel. plot

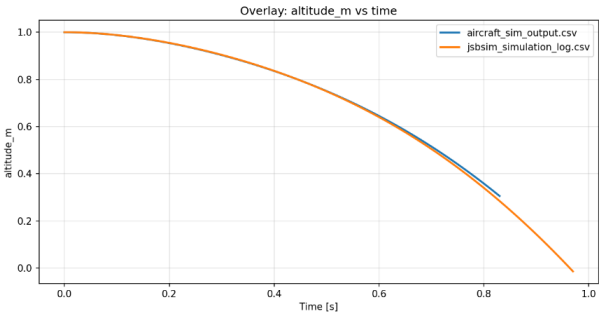
(a) Shared-column error statistics and a 1 m no-actuation altitude overlay (ground effect disabled).

Shared-column Comparison: aircraft_sim_output.csv vs jsbsim_simulation_log.csv

column	count	bias_mean	mae	rmse	max_abs_error	corr
time_s	3628	0	0	0	0	1
theta_rad	3628	0.00192849	0.00763397	0.00870776	0.0180156	0.99754
q_rad_s	3628	0.000499503	0.00318061	0.00410824	0.0180145	0.996867
u_m_s	3628	-0.239356	0.263484	0.322083	0.612385	0.997333
altitude_m	3628	1.38267	1.38267	1.59961	2.92249	0.999551

Analysis Methods

(a) - 100m initial altitude, no actuation - Statistics



(b) - 1m initial altitude, no actuation - altitude plot - No GE

(b) Altitude and pitch-rate (q) overlays from a 100 m no-actuation free-flight episode.

Figure 21: Cross-validation of the in-house simulator against JSBSim under identical zero-input free-flight conditions. Blue: in-house simulator; orange: JSBSim. Altitude and pitch-rate correlations exceed 0.99 throughout.

Trajectories that agree across all three engines are promoted to the hardware-in-the-loop stage; those that diverge flag a modelling gap to be resolved before any hardware is involved.

5.5 Hardware-in-the-loop on the ironbird

As a final verification step, the learned and classical controllers are exercised on an *ironbird*: a fixed bench carrying the actual flight computer, sensors, actuators and propulsion electronics in their flight configuration, with the airframe absent so that nothing flies (Fig. 22). The bench reproduces the full avionics chain — the Cube autopilot (CubePilot, 2024) (to be later substitudes with the A0S Pixhawck 4), GPS/receiver and antennas, the PWM driver, an RC multiplexer for pilot/autopilot handover, and the servo/BEC power path. The simulator streams the vehicle state to the flight computer, the onboard controller computes commands through the same stack of Fig. 13, and the physical actuators move exactly as they would in flight, so wiring, timing, signal scaling, the multiplexer handover logic and actuator behaviour are all exercised on real hardware without flight risk. This bench complements the A0-S HIL campaign of Section 4 by isolating the controller–avionics interface under repeatable, fully simulated physics.

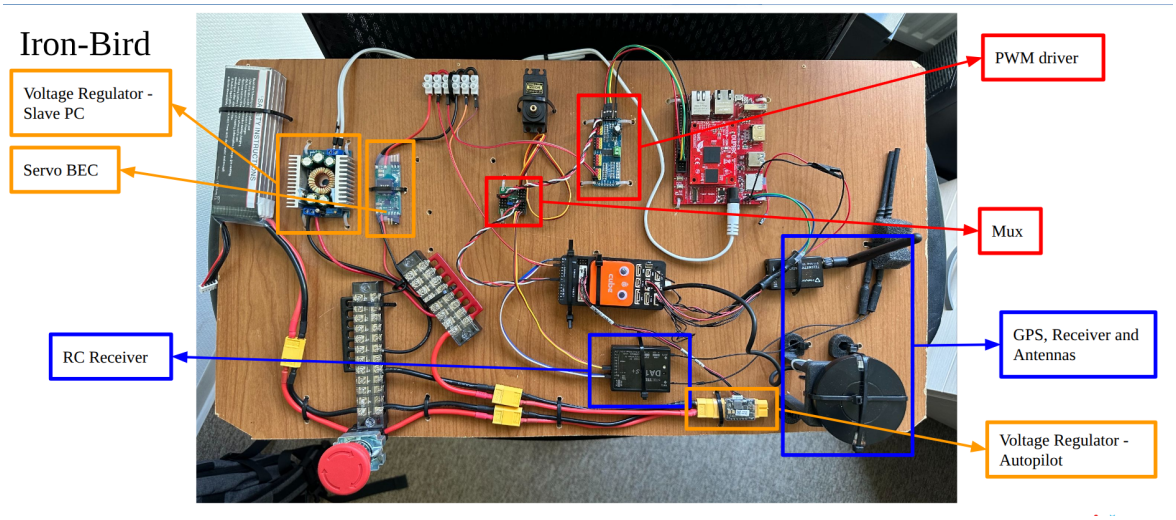


Figure 22: The ironbird bench: the AIRSHIP avionics (Cube autopilot, PWM driver, RC multiplexer, GPS/receiver and antennas, voltage regulators and servo BEC) mounted in flight configuration with the airframe absent, driven in real time by the simulator over the hardware-in-the-loop link.

6 Conclusions and Future Work

Task 5.3 has established the simulation and validation infrastructure required for the progressive development of the AIRSHIP autonomous system. Through the development of a comprehensive digital twin together with SIL and HIL frameworks, the project has created a unified environment for the design, integration, and verification of software and hardware components before deployment on the physical prototypes.

The digital twin, continuously refined throughout the project as the AIRSHIP vehicle design evolved, provides realistic representations of the flight dynamics, onboard sensors, and operational environment. Coupling X-Plane 12 with Gazebo has enabled high-fidelity simulation of both vehicle behaviour and perception systems, supporting the development and validation of guidance, navigation, control, perception, and mission management algorithms.

The SIL framework has enabled extensive validation of the AIRSHIP software stack under representative operating conditions, while the HIL framework has progressively introduced real onboard hardware into the simulation loop, culminating in validation activities on the A0-S prototype. Together, these frameworks have significantly reduced the technical risks associated with software integration and autonomous flight testing by identifying integration issues at an early stage of development.

Beyond supporting the control system development performed in Task 5.2, the validation infrastructure established in Task 5.3 provides a common experimental platform for Task 6.4 – Situational Awareness and Task 7.1 – Guidance and Navigation. The developed environments support the validation of environment perception, sensor fusion, situational awareness, guidance, trajectory planning, mission execution, and obstacle avoidance algorithms under realistic maritime operating conditions.

Future work will focus on extending the HIL validation activities to increasingly autonomous flight experiments, further improving the fidelity of the digital twin, and expanding the simulated operational scenarios to support advanced autonomy functions. The developed validation framework will continue to support the integration and verification activities leading to the experimental demonstrations planned in WP8, providing a robust bridge between simulation, laboratory validation, and real-world deployment of the AIRSHIP WIG vehicle.

Funded by
the European Union

References

Aravind, S. (2024). *gazebo_terrain_generator: A tool for generating gazebo terrain using satellite images and dem data.* https://github.com/saiaravind19/gazebo_terrain_generator. (Version 2.0, BSD-3-Clause License)

Berndt, J. S., & the JSBSim Development Team. (2011). *JSBSim: An Open Source Flight Dynamics Model in C++.*

Bøhn, E., Coates, E. M., Moe, S., & Johansen, T. A. (2019). Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. *CoRR*, *abs/1911.05478*. Retrieved from http://arxiv.org/abs/1911.05478

CubePilot. (2024). *Cube orange+ store page.* Retrieved from https://www.cubepilot.com/#/cube/features

Ermakov, V., & Community, M. (2024). *Mavros: Mavlink extendable communication node for ros.* https://github.com/mavlink/mavros.

Kwakernaak, H., & Sivan, R. (1972). *Linear optimal control systems*. Wiley-Interscience.

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

NASA. (2017). *nasa/XPlaneConnect: The X-Plane Communications Toolbox.* Retrieved from https://github.com/nasa/XPlaneConnect/wiki

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, *22*(268), 1-8. Retrieved from http://jmlr.org/papers/v22/20-1364.html

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., … others (2024). Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032*.