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

D5.2 Report on results of control system on physical model

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)	T5.2
Lead Organisation	Luxemburg University
Contributing Partner(s)	UPM, TAU, INESC TEC
Due Date	31/12/2025
Submission Date	31/12/2025
Dissemination Level	PU

Revision History

Revision	Revision Date	Name	Position
1.0	19/12/2025	Pedro Lima (UL)	Researcher
2.0	23/12/2025	Zorana Milosevic (UPM)	Researcher
3.0	30/12/2025	Josep Sánchez (TRISOLARIS)	Researcher

TABLE OF CONTENTS

1 Introduction 10
1.1 Purpose of the Deliverable . . . 10
1.2 Document Structure . . . 10

2 Preliminaries 12
2.1 Aerodynamics forces and moments . . . 14
2.1.1 Aerodynamic coefficients estimation . . . 14
2.1.2 Ground effect modeling . . . 15
2.2 Propulsion forces and moments . . . 15
2.3 Actuator dynamics . . . 16

3 Algorithmic control techniques 17
3.1 PID Control . . . 17
3.1.1 PID control architecture . . . 17
3.1.2 Waypoint Tracking Guidance . . . 22
3.1.3 Experimental Results . . . 23
3.1.4 Time-Domain Performance Metrics . . . 26
3.1.5 Tracking Performance Metrics . . . 28
3.2 Linear Quadratic Gaussian (LQG) Control . . . 30
3.2.1 Trimming the WIG Vehicle . . . 31
3.2.2 Linearization around operating trim point . . . 32
3.2.3 Simulation results . . . 32
3.3 Linear Quadratic Regulator (LQR) Control . . . 34
3.3.1 Linear Quadratic Regulator with Integral action . . . 35
3.3.2 LQRI simulation results . . . 37
3.4 $\mathcal{L}_1$ Adaptive Control . . . 39
3.4.1 Baseline Controller . . . 40
3.4.2 $\mathcal{L}_1$ Adaptive Controller . . . 41
3.4.3 Simulation experiments . . . 41
3.4.4 Monte Carlo Robustness Analysis . . . 42
3.4.5 Results . . . 42
3.5 Fuzzy Control . . . 45
3.5.1 Controller Design . . . 46
3.5.2 Implementation . . . 47
3.5.3 Results, Evaluation and Conclusion . . . 48

4 AI-based control 51
4.1 Learning-Based Control with DreamerV3 . . . 51

5 Simulation frameworks 52
5.1 Python framework . . . 53
5.2 MATLAB/Simulink framework . . . 55
5.3 X-Plane 12 framework . . . 55
5.3.1 System Architecture . . . 56

6 Control Software Architecture 59
6.1 Development . . . 59

6.1.1 System Requirements . . . 60
6.1.2 Hardware Architecture . . . 60
6.1.3 Software Architecture Design . . . 61
6.2 Experimental Evaluation and Results . . . 64
6.2.1 Discussion on the results . . . 65

7 Conclusion and Future Work 67

References 68

List of Tables

Table 1 Abbreviation and Acronyms . . . 7
Table 2 List of Symbols . . . 8
Table 3 List of System Parameters . . . 8
Table 4 Dryden turbulence velocity spectral filters. . . . 13
Table 5 Time-domain performance metrics . . . 29
Table 6 Tracking performance metrics . . . 30
Table 7 Mean squared error (MSE) for different noise levels . . . 33
Table 8 Summary of evaluation metrics computed from trajectory and control logs. . . . 37
Table 9 Performance metrics (RMSE, settling time, σ, overshoot, mean error, and energy utilization) for the AIRSHIP vehicle in tasks (a)–(f) with the LQRI controller. . . . 37
Table 10 LQRI performance metrics (RMSE, settling time, overshoot, error mean + std, and energy utilization) for the AIRSHIP vehicle at various constant altitudes. . . . 38
Table 11 Mean ± standard deviation across five DreamerV3 runs for dynamic altitude tracking. . . . 53
Table 12 Average CPU Load in Total and Per Core at Different Command Update Rates . . . 65
Table 13 Average Network Throughput (Sent and Received) and Memory Used at Different Command Update Rates . . . 65

List of Figures

Figure 1 Pitch control loop. . . . 19
Figure 2 Throttle control loop. . . . 19
Figure 3 Heading control loop. . . . 20
Figure 4 **Cross-Track Error (XTE) calculation**. A part of the actual path is shown in this figure. The perpendicular distance of a position of the WIG craft from the ideal straight line trajectory is known as the Cross-track distance. This error is then minimized by roll angle inputs. . . . 21
Figure 5 Roll control loop. . . . 22
Figure 6 Complete guidance flowchart. . . . 22
Figure 7 2D ”square” flight path. . . . 24
Figure 8 2D ”eight” flight path. . . . 25
Figure 9 2D ”eight” altitude control output. . . . 25
Figure 10 3D flight path (Top view). . . . 26
Figure 11 3D flight path. The axes represent latitude and longitude of the waypoints. The mission starts at GPS coordinate $(40.40°, -18.40°)$, a region of the North Atlantic Ocean. The WIG craft is tasked to simultaneously control position and height. . . . 27
Figure 12 3D control outputs for altitude, pitch, roll, and yaw. . . . 28
Figure 13 LQG Block Scheme . . . 31

Figure 14 LQG control results for velocity and attitude-related states. . . . 33
Figure 15 LQG control results for velocity and flight path angle. . . . 33
Figure 16 LQG control results. . . . 34
Figure 17 LQRI controller block diagram. . . . 36
Figure 18 Comparison of tracking performance for LQR and LQRI control laws. The LQR response exhibits a non-zero steady-state error, while LQRI achieves zero steady-state error due to integral action. . . . 36
Figure 19 LQRI-controlled AIRSHIP response to tracking different trajectories. . . . 38
Figure 20 Variation of C_L and C_D with altitude due to ground effect, utilizing LQRI for constant altitude tracking. . . . 39
Figure 21 Monte Carlo simulation results (20 runs) showing control inputs. Colored lines correspond to Monte Carlo realizations with parameter uncertainty, while the black line represents the nominal case. . . . 43
Figure 22 Monte Carlo simulation results (20 runs) showing key state variables: airspeed, angle of attack, pitch angle, and altitude. Colored trajectories correspond to perturbed parameter cases, while the black trajectory represents the nominal model. . . . 44
Figure 23 Zoomed view of altitude response during the initial ground-effect flight phase. The figure highlights controller behavior near the water surface, where aerodynamic sensitivity is highest. . . . 44
Figure 24 Membership Functions . . . 46
Figure 25 Custom Graphical User Interface . . . 48
Figure 26 X-Plane 12 Simulation . . . 49
Figure 27 Fuzzy control performance during takeoff. . . . 50
Figure 28 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. . . . 52
Figure 29 Illustration of different sensor effects implemented in the Python simulator. . . . 54
Figure 30 Block-diagram representation of the implemented aircraft dynamics and control architecture in Simulink, illustrating the modular structure of the simulation framework, with separate subsystems for actuator dynamics, sensor models, vehicle dynamics and feedback control laws. . . . 55
Figure 31 AIRSHIP prototype designed in PlaneMaker, a proprietary software that comes with X-Plane 12. . . . 56
Figure 32 Prototype tested in X-Plane 12. . . . 57
Figure 33 ROS 2 - X-Plane 12 closed-loop control architecture. . . . 58
Figure 34 Top view of the AIG printed circuit board (blue) populated with the RK3328 System-on-Module (red), used for real-time embedded control. . . . 61
Figure 35 High-Level SW architecture diagram. The flight controller manages sensors, telemetry, and control commands, while the AIG-API layer provides ROS2/MAVROS middleware and Unix socket services for motors, servos, and auxiliary systems. . . . 63
Figure 36 Low-level SW architecture within AIG-API. ROS2 nodes handle sensor logging, monitoring, and control, Unix sockets ensure direct, reliable actuation of motors, servos, and navigation lights. The diagram reflects the actual ROS2 topic structure. . . . 64
Figure 37 Average CPU Load at Different Command Update Rates . . . 65

Abbreviations and Acronyms

Acronym	Description
AI	Artificial Intelligence
CPU	Central processing unit
DoF	Degrees-of-freedom
GNC	Guidance, Navigation, and Control
GNSS	Global Navigation Satellite System
GPU	Graphics processing unit
HIL	Hardware-In-the-Loop
IMU	Inertial Measurement Unit
ISA	International Standard Atmosphere
ISO	International Organization for Standardization
LQR	Linear Quadratic Regulator
LQRI	Linear Quadratic Regulator with Integral action
LSTM	Long Short-Term Memory
LTI	Linear time-invariant
NED	North-East-Down reference frame
ODE	Ordinary Differential Equation
PID	Proportional–Integral–Derivative controller
PPO	Proximal policy optimization
RL	Reinforcement Learning
RSSM	Recurrent state-space model
SAC	Soft Actor Critic
SIL	Software-In-the-Loop
UAV	Unmanned Aerial Vehicle

UWV	Unmanned WIG Vehicle
WIG	Wing-in-ground

Table 1: Abbreviation and Acronyms

List of Symbols

Symbol	Description
α	Angle of attack
β	Side-slip angle
$\boldsymbol{v}$	Velocity vector
$\boldsymbol{\omega}$	Angular velocity vector
$\boldsymbol{q}$	Attitude quaternion
$\boldsymbol{p}$	Position vector
$\boldsymbol{R}_b^I$	Rotation matrix from the frames b to I
$\boldsymbol{g}$	Gravity vector
$\boldsymbol{h}$	Altitude
$\boldsymbol{F}$	Force vector
L	Lift
$\boldsymbol{M}$	Moment vector
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{S}$	Skew-symmetric matrix
S	Wingspan
t	Time
T	Temperature
$G(s)$	Transfer function
τ	Time constant

ζ	Damping ration
ω_n	Natural Frequency
p_t	Total pressure
p_s	Static pressure

Table 2: List of Symbols

List of System Parameters

Symbol	Description	Value
ρ	Air Density at sea level	$1.225\ kg.m^{-3}$
g	Gravity acceleration	$9.81\ m.s^{-2}$
b	Wingspan	$5.0\ m$
c	Wing cord	$0.62\ m$
AR	Aspect ratio	7.53
λ	Tamper ratio	0.39

Table 3: List of System Parameters

Executive Summary

This deliverable reports the work carried out under Task 5.2 - WIG 6 DOF control ([M12-M36]), led by UPM with participation of LU, TAU, and INESC. It presents the research, development, and evaluation of control strategies for the AIRSHIP A-1 prototype, aimed at achieving autonomous stabilization and reference tracking of flight trajectories within ground-effect conditions, while ensuring both robustness and safety across the relevant flight envelope.

Task 5.2 builds upon the initial dynamic model developed in Task 5.1 ([M1-M12]). While this early model provided a necessary baseline, it was derived at a stage when the AIRSHIP vehicle design was still at a conceptual level. Consequently, the model has been significantly refined and extended during Task 5.2, incorporating updated aerodynamic, structural, and propulsion characteristics as the vehicle definition matured. These improvements were essential to enable meaningful controller design and performance assessment.

The work in Task 5.2 is structured according to the two subtasks defined:

- Subtask 5.2.1: Algorithmic control techniques, where a comprehensive analysis of the linear and nonlinear control methodologies is investigated.
- Subtask 5.2.2: AI-based control techniques, which investigated the use of reinforcement learning–based approaches.

Task 5.2 is tightly coupled with Task 5.3, which provides the simulation and Software-in-the-Loop (SIL) environments used for controller development, verification, and sim-to-real validation under Task 5.2. In addition, the control architectures and low-level stabilization strategies developed in Task 5.2 directly support Task 7.1 – Guidance and Navigation, by providing the reliable low-level control layer required for higher-level guidance functions such as waypoint following, trajectory execution, and maneuver implementation. Finally, the deliverable describes the planned next steps toward Hardware-in-the-Loop (HIL) integration and flight testing.

1 Introduction

The present deliverable focuses on the research, design, and implementation of various control strategies for the AIRSHIP project, whose overarching goal is to develop an innovative, environmentally friendly wing-in-ground (WIG) effect vehicle capable of safe, efficient, and autonomous operation. Within this broader context, the work presented in this report specifically addresses one of the central technological challenges that have historically limited the widespread adoption of WIG vehicles for cargo and passenger transport, their stability and control under flight conditions close to the surface (Amir, Maimun, Mat, Saad, & Zarim, 2016). The aerodynamic environment in ground effect is characterized by complex nonlinear interactions, strong coupling between dynamic states, and sensitivity to attitude and altitude variations (Phillips & Hunsaker, 2013). These phenomena introduce significant challenges for both modeling and control, requiring robust, adaptive, or learning-based control approaches capable of ensuring stability, robust trajectory tracking, and safety across the full operational envelope.

To tackle these limitations, this report details the development and evaluation of both algorithmic and learning-based control techniques for the AIRSHIP A1 vehicle. The first part of the work focuses on deterministic, model-based approaches, including linear and nonlinear control strategies. These deterministic techniques are well-established within the aerospace industry and form the foundation of many modern autopilot systems, such as ArduPilot and PX4, due to their proven reliability, interpretability, and ease of implementation. In this context, they will serve not only as a performance baseline, but also as a reference point for assessing the effectiveness and practical value of the more unconventional RL-based control approaches.

By researching both traditional control theory and RL-based methodologies, this deliverable aims to establish the groundwork for the autonomous Guidance, Navigation, and Control (GNC) system of the AIRSHIP platform. The outcomes of this work aim to contribute to the project's broader objectives of validating autonomous operation in ground-effect flight through simulation, HIL testing, and subsequent real-world experimentation utilizing small-scale prototypes.

1.1 Purpose of the Deliverable

The purpose of this deliverable is to report the work performed under Task 5.2 - WIG 6-DOF Control. It presents the ongoing research, development, and preliminary validation of the control strategies for the AIRSHIP A1 vehicle, aimed at achieving robust, safe and autonomous flight within ground-effect conditions. This document consolidates the theoretical background, implementation methodologies, and experimental results associated with both algorithmic and learning-based approaches for control. The reported work builds upon the dynamic modeling activities of Task 5.1 and is directly supported by the simulation and SIL environments developed under Task 5.3. In addition, the low-level stabilization and control architectures developed in Task 5.2 provide the foundational control layer required for the higher-level Guidance and Navigation functions addressed in Task 7.1, including waypoint following and trajectory execution. The results presented in this deliverable establish the control foundation required for subsequent HIL integration and experimental validation.

1.2 Document Structure

This document is organized as follows:

- Section 2 - Introduces the theoretical foundations and system modeling aspects relevant to the

control of the AIRSHIP A1 vehicle, including key aerodynamic characteristics and ground-effect considerations that influence controller design.

- Section 3 - Presents the algorithmic control approaches investigated in this work, such as cascade PID, LQG, LQR, L1, and Fuzzy Control.
- Section 4 - Explores the use of AI-based control, in particular Reinforcement Learning algorithms, such as DreamerV3, for achieving autonomous control of the vehicle in ground-effect conditions. The section discusses training procedures, reward design, and policy evaluation.
- Section 5 - Describes the simulation environments and tools employed for development and validation, including Python-based frameworks using NVIDIA Warp, as well as MATLAB/Simulink and X-Plane environments.
- Section 6 - Describes the software architecture developed for integration of control algorithms to the AIRSHIP prototype.
- Section 7 - Summarizes the main findings, highlights the performance of the different control approaches, and outlines the planned next steps toward hardware-in-the-loop testing and full flight validation.

2 Preliminaries

In order to develop an accurate, custom physics-based simulation framework and model-based controllers, a mathematical description of the vehicle has to be developed. It is considered that the vehicle is a rigid body fixed-wing craft flying in proximity to the ground, modeled as a six-degrees-of-freedom (6DoF) system with coupled translational and rotational dynamics.

The vehicle is set to have a constant mass m with inertia tensor $\boldsymbol{J}$, linear velocity $\boldsymbol{v} = [u\, v\, w]^T$ and angular velocity $\boldsymbol{\omega} = [p\, q\, r]^T$. A fixed body frame $\{B\}$ is defined at the centre of gravity, moving and rotating relative to a fixed north-east-down (NED) inertial frame $\{I\}$. The attitude is represented using unit quaternion $\mathsf{q} \in S^3 = \{\mathsf{q} \in \mathbb{H} : ||\mathsf{q}|| = 1\}$, where $\mathbb{H}$ represents the set of all quaternions. The position $\boldsymbol{p} = [x\, y\, z]^T$ and 6DoF attitude kinematics are govern by:

$$\dot{\boldsymbol{p}} = \boldsymbol{R}_b^I(\boldsymbol{q})\, \boldsymbol{v}, \tag{1}$$

$$\dot{\boldsymbol{q}} = \frac{1}{2} \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & -\boldsymbol{S}(\boldsymbol{\omega}) \end{bmatrix} \boldsymbol{q}, \tag{2}$$

where $\boldsymbol{R}_b^I(\boldsymbol{q})$ is the rotation matrix that rotates vectors from the body frame $\{B\}$ to the inertial frame $\{I\}$ using the unit quaternion $\boldsymbol{q}$, and $\boldsymbol{S}(\boldsymbol{\omega})$ is simply represents the skew-symmetric matrix of $\boldsymbol{\omega}$.

The system dynamics, represented in the fixed inertial frame $\{I\}$ as

$$\frac{d}{dt}[m\boldsymbol{v}]_I = \boldsymbol{F}, \tag{3}$$

$$\frac{d}{dt}[\boldsymbol{J}\boldsymbol{\omega}]_I = \boldsymbol{M}, \tag{4}$$

with $\boldsymbol{F}$ and $\boldsymbol{M}$ representing, respectively, the total external forces and moments acting on the WIG vehicle. In the non-inertial frame of reference $\{B\}$, the dynamics are

$$\frac{d}{dt}[m\boldsymbol{v}]_B + \boldsymbol{\omega} \times [m\boldsymbol{v}]_B = \boldsymbol{F}, \tag{5}$$

$$\frac{d}{dt}[\boldsymbol{I}\boldsymbol{\omega}]_B + \boldsymbol{\omega} \times [\boldsymbol{I}\boldsymbol{\omega}]_B = \boldsymbol{M}. \tag{6}$$

The external forces considered (relative to $\{B\}$) are: the force due to gravity $\boldsymbol{F}_g = \boldsymbol{R}_b^I(\boldsymbol{q})^T\, m\boldsymbol{g}$, with $\boldsymbol{g}$ being the gravity acceleration $\boldsymbol{g} = [0\, 0\, g]^T$, the force due to aerodynamics $\boldsymbol{F}_{aero}$, and the propulsion force $\boldsymbol{F}_P$. The external moments are also considered to be exerted by the aerodynamics $\boldsymbol{M}_{aero}$ and propulsion $\boldsymbol{M}_P$ (see sections 2.1 and 2.2).

Lastly, the vehicle is considered to be flying in a wind field composed of a steady disturbance $\boldsymbol{v}_{w_s}$, and a stochastic *Dryden Turbulence* gusts $\boldsymbol{v}_{w_d}$, meaning that the velocity of the vehicle relative to the wind field is given by:

$$\boldsymbol{v}_r = \boldsymbol{v} - \boldsymbol{R}_b^I(\boldsymbol{q})^T \boldsymbol{v}_{w_d} - \boldsymbol{v}_{w_s} = [u_r\ v_r\ w_r]^T. \tag{7}$$

The Dryden turbulence (Dryden & Kuethe, 1930) was implemented via the *MIL-F-8785C* model using band-limited white noise through forming low-pass filters (see table 4). For low altitude flights ($h < 1000ft$), the turbulence scale lengths and intensities are defined as

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

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.

Table 4: Dryden turbulence velocity spectral filters.

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

The airspeed V_a, angle of attack α and side-slip angle β can be derived from relative velocity $\boldsymbol{v}_r$:

$$V_a = \|\boldsymbol{v}_r\|, \tag{10}$$

$$\alpha = \arctan\left(\frac{w_r}{u_r}\right), \tag{11}$$

$$\beta = \arcsin\left(\frac{v_r}{V_a}\right). \tag{12}$$

The air density altitude gradient is computed following the 1975 International Standard Atmosphere (ISA) (ISO 2533:1975 (*ISO 2533:1975 Standard Atmosphere*, 1975)) for altitudes below $h < 11000m$:

$$T = T_0 - Lh \tag{13}$$

$$p = p_0\left(\frac{T}{T_0}\right)^{\left(\frac{g}{RL}\right)} \tag{14}$$

$$\rho = \frac{p}{RT} \tag{15}$$

where $T_0 = 288.15\ K$ and $p_0 = 101325.0\ Pa$ are the sea level standard temperature and pressure respectively, $L = 0.0065\ K \cdot m^{-1}$ the temperature lapse rate, $g = 9.80665\ m \cdot s^{-1}$ the acceleration due to gravity and $R = 287.05287\ J \cdot kg^{-1} \cdot K^{-1}$ specific gas constant for dry air.

2.1 Aerodynamics forces and moments

The aerodynamic forces $\boldsymbol{F}_{aero}$ and moments $\boldsymbol{M}_{aero}$ are formulated in terms of the non-linear aerodynamic coefficients and are given by

$$\boldsymbol{F}_{aero} = \boldsymbol{R}_w^b\left(\alpha, \beta\right) \begin{bmatrix} -D \\ Y \\ -L \end{bmatrix}, \tag{16}$$

$$\begin{bmatrix} D \\ Y \\ L \end{bmatrix} = \frac{1}{2}\rho V_a^2 S \begin{bmatrix} C_D\left(\alpha, \beta, \delta_a, \delta_e, \delta_r, Re, M\right) \\ C_Y\left(\alpha, \beta, \delta_a, \delta_r, Re, M\right) \\ C_L\left(\alpha, \beta, \delta_e, Re, M\right) \end{bmatrix}, \tag{17}$$

$$\boldsymbol{M}_{aero} = \frac{1}{2}\rho V_a^2 S \begin{bmatrix} bC_l\left(\beta, p, r, \delta_a, \delta_r, Re, M\right) \\ cC_m\left(\alpha, q, \delta_e, Re, M\right) \\ bC_n\left(\beta, p, r, \delta_a, \delta_r, Re, M\right) \end{bmatrix}, \tag{18}$$

where ρ is the air density, S is the projected wing area, b is the wingspan, c is the wing cord, Re is the *Reynolds* number and M is the *Mach* number. The aileron, elevator and rudder angle deflections are given by δ_a, δ_e and δ_r respectively.

The rotation matrix $\boldsymbol{R}_w^b(\alpha, \beta)$ from the wind frame to the body frame is given by:

$$\boldsymbol{R}_w^b\left(\alpha, \beta\right) = \begin{bmatrix} \cos\left(\alpha\right)\cos\left(\beta\right) & \cos\left(\alpha\right)\sin\left(\beta\right) & -\sin\left(\alpha\right) \\ -\sin\left(\beta\right) & \cos\left(\beta\right) & 0 \\ \sin\left(\alpha\right)\cos\left(\beta\right) & \sin\left(\alpha\right)\sin\left(\beta\right) & \cos\left(\alpha\right) \end{bmatrix}. \tag{19}$$

2.1.1 Aerodynamic coefficients estimation

To obtain aerodynamic coefficient models for the flight simulations, a CAD representation of the AIRSHIP vehicle was imported into *OpenVSP* (NASA OpenVSP Team, 2025). A custom automation script was developed to perform systematic aerodynamic sweeps at fixed Reynolds and Mach numbers. The simulations were conducted over ranges of angle of attack α, side-slip angle β, and control surface (δ_a, δ_e and δ_r). The resulting data were organized into lookup tables expressing the aerodynamic coefficients as functions of the primary kinematic and control variables, namely $C_D(\alpha, \delta_e)$, $C_Y(\beta, \delta_r)$, $C_L(\alpha, \delta_e)$, $C_l(\beta, \delta_a)$, $C_m(\alpha, \delta_e)$, and $C_n(\beta, \delta_r)$. Subsequently, two-dimensional third-degree linear regression models of the form

$$f\left(x, y\right) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 + a_6 x^3 + a_7 x^2 y + a_8 xy^2 + a_9 y^3, \tag{20}$$

were fitted to the generated datasets for each aerodynamic coefficient to accelerate computations and reduce memory allocation.

2.1.2 Ground effect modeling

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

$$C_L = C_L^{\infty} \left(1 + \mu_L \left(h/b\right)\right), \tag{21}$$

$$C_D = C_D^{\infty} \left(1 - \mu_D \left(h/b\right)\right) \left(1 + \mu_L \left(h/b\right)\right)^2, \tag{22}$$

where μ_L, μ_D are ground effect modifiers parametrized as functions of the height-wingspan ratio (h/b) given by:

$$\mu_L = \delta_L \frac{288 \left(h/b\right)^{0.787} \exp\left[-9.14 \left(h/b\right)^{0.327}\right]}{AR^{0.882}}, \tag{23}$$

$$\delta_L = 1 - 2.25 \left(\lambda^{0.00273} - 0.997\right) \left(AR^{0.717} + 13.6\right), \tag{24}$$

$$\mu_D = \delta_D \exp\left[-4.74 \left(h/b\right)^{0.814}\right] + \left(h/b\right)^2 \exp\left[3.88 \left(h/b\right)^{0.758}\right], \tag{25}$$

$$\delta_D = 1 - 0.157 \left(\lambda^{0.775} - 0.373\right) \left(AR^{0.417} - 1.27\right). \tag{26}$$

which assumes an untwisted linear tapered wing at low angles of attack, with $\lambda = c_{tip}/c_{root}$ representing the taper ratio, AR the wing aspect ratio, and C_L^{∞} and C_D^{∞} denoting the estimated out-of-ground-effect coefficients.

2.2 Propulsion forces and moments

The thrust force is generated by the thrust from two propellers:

$$\boldsymbol{F}_P = \boldsymbol{n}_{P_1} T_{prop_1} + \boldsymbol{n}_{P_2} T_{prop_2} \tag{27}$$

with $\boldsymbol{n}_P$ being the normalized thrust direction vector.

The propeller thrust T_{prop} is approximated by (Bøhn, Coates, Moe, & Johansen, 2019):

$$V_d = V_a + \delta_T \left(k_m - V_a\right) \tag{28}$$

$$T_{prop} = \frac{1}{2} \rho S_P C_P V_d \left(V_d - V_a\right) \tag{29}$$

where V_d is the discharge velocity of air from the propeller, k_m is the electric motor constant, S_p is the propeller disc area, C_p is an efficiency factor, and $\delta_T \in [0, 1]$ is the throttle.

The propulsion-generated moment is composed of the moment generated by the thrust force acting at an offset from the centre of gravity and the additional moment contribution due to propeller torque:

$$\boldsymbol{M}_P = \boldsymbol{p}_{motor_1} \times \boldsymbol{F}_{P_1} + \boldsymbol{M}_{prop_1} + \boldsymbol{p}_{motor_2} \times \boldsymbol{F}_{P_2} + \boldsymbol{M}_{prop_2}, \tag{30}$$

where $\boldsymbol{p}_{motor}$ is the position vector of each motor in the body frame and $\boldsymbol{M}_{prop}$ is the propeller-generated torque. Assuming both propellers rotate at the same angular velocity in opposite directions, then $\boldsymbol{M}_{prop} = 0$.

Gyroscopic moments are also assumed to be negligible.

2.3 Actuator dynamics

To model the throttle and surface deflection dynamics, first and second-order transfer functions with saturation approximations were used, respectively:

$$\frac{\delta_t(s)}{\delta_t^c(s)} = \frac{1}{Ts+1}, \tag{31}$$

$$\frac{\delta_i(s)}{\delta_i^c(s)} = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \tag{32}$$

for $i = a, e, r, \omega_0 = 10, \zeta = \frac{\sqrt{2}}{2}$ and $T = 0.2$, with the superscript c denoting the controller commanded actions; following the same parameters choices as in (Bøhn et al., 2019).

3 Algorithmic control techniques

This section presents a set of representative control strategies investigated within Task 5.2 for the autonomous operation of the AIRSHIP vehicle. The developed controllers span classical, optimal, adaptive, and rule-based control paradigms, allowing a structured assessment of their suitability for low-altitude, ground-effect flight characterized by nonlinear aerodynamics, strong coupling, and parametric uncertainty.

Each subsection describes the controller formulation, implementation details, and corresponding simulation results obtained. The controllers are not intended as mutually exclusive solutions, but rather as complementary approaches that establish performance baselines, explore robustness to uncertainties, and inform design trade-offs for subsequent hardware-in-the-loop and experimental validation phases.

3.1 PID Control

Despite the increasing adoption of advanced control techniques in aerospace systems, PID-based control architectures remain the foundation of many widely used off-the-shelf autopilots, such as ArduPilot, PX4, and NAVIO2. These systems are commonly based on cascaded proportional–integral–derivative (PID) loops, employing independent controllers for roll, pitch, and yaw, and typically operating under the assumption of weakly coupled or decoupled longitudinal and lateral dynamics (Beard & McLain, 2012) (Kopečnỳ, Hnidka, & Bajer, 2022).

For this reason, the control development within Task 5.2 begins with PID control. Establishing a well-tuned PID baseline provides a familiar and reliable reference point from both an industrial and research perspective, while enabling an initial assessment of the WIG vehicle's dynamic behavior in ground-effect conditions. In particular, PID control facilitates early evaluation of control bandwidth, sensitivity to aerodynamic uncertainties, and the impact of ground-effect-induced nonlinearities before introducing more complex control structures.

In the context of WIG vehicle guidance, independent PID loops can be employed to regulate both longitudinal and lateral motion. While this approach does not explicitly address the strong coupling and nonlinearities inherent to ground-effect flight, it provides valuable insight into the achievable performance and inherent limitations when using classical control techniques.

The remainder of the section outlines the developed PID control architecture and presents the experimental results. The work presented in this section has been submitted to the IEEE Access journal under the title "Low-Altitude Waypoint Tracking Control of a Wing-in-Ground-Effect Vehicle".

3.1.1 PID control architecture

The control architecture implemented is based on cascaded PID controllers. Specifically, it combines a PI–PID cascaded loop for pitch control, a PI throttle loop for energy management, a PID-based heading controller for lateral guidance, and a cascaded PD–PI roll controller driven by cross-track error, with roll commands explicitly bounded to ensure safe low-altitude operation.

The standard PID controller in the continuous time domain is

$$u(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt} \tag{33}$$

where K_p, K_i, K_d are proportional, integral and derivative gain constants, respectively. Thus, the PID controller consists of three components: proportional, integral, and differential, each of which serves a specific purpose. The proportional component generates a command control signal that counteracts the deviation of the intended output signal, also known as the set point. As a result, it is tasked with directly reducing the error signal. If the commanded output signal equals the setpoint, the error is zero, and the proportional component's commanded control action is zero. The integral component eliminates steady-state error with respect to the reference value. The differential component considers the rate of change of the output signal, which provides greater control over the object by forecasting the output value of the signal (*Basics of PID Control*, 2006).

A PID controller for waypoint tracking computes the control commands that guide the vehicle across a sequence of waypoints by minimizing the error between the current and the desired attitude estimates. In such applications, the controller continuously adjusts the vehicle's position and heading through commands generated for actuators, such as control surface deflections and throttle outputs. A series of PID loops have been implemented, which were tuned for optimal behavior, to achieve waypoint tracking.

The design of PID control loops has been carried out with consideration for the task at hand, following a step-by-step iterative process. Each of the axes has been kept stable and decoupled from the others during the tuning process. All the control architecture diagrams have been displayed in the subsequent sections. Final tuning has been carried out with the assistance of a live tuner developed within the ROS 2 framework. This tuner is equipped with a live parameter change feature that enables the controller to alter the gains even when the simulation is running. This feature is essential when working with complex systems that involve multiple loops, such as this one.

Longitudinal Control

Longitudinal Control - Pitch

The longitudinal control of a WIG craft involves controlling the craft's altitude. The altitude is maintained with the help of the pitch and the throttle output of the craft. The pitch of any fixed-wing aircraft is controlled with the use of the elevator control surface and hence the same for our WIG vehicle. Some aircraft have multiple elevators while others have a single elevator. Our WIG craft has a single elevator control surface.

Hence, the altitude of the WIG craft is controlled through a cascaded PID loop structure as shown in Fig. 1. Since the vehicle is flying close to the surface, a maximum pitch angle limit of $\pm 15.0^{\circ}$ is set. This was done keeping the rest of the axes, such as the roll axis and yaw axis, stable and decoupled. The outer loop of the cascaded PID consists of an altitude controller, which takes into account the height error and outputs a desired pitch angle. The outer loop contains PID terms. The inner loop, which in our case is a PI controller, then inputs the desired pitch angle command and converts it into an elevator command value between -1 and 1. The elevator command is then sent to the X-Plane 12 simulator.

Longitudinal Control - Throttle

The second component of a longitudinal control autopilot is the throttle control. The altitude of a WIG craft cannot be maintained only with the use of elevator control surfaces. Several studies in the literature demonstrate the suitability of using an altitude-based PID controller for altitude hold tasks (Mazlan, Shamsudin, Pairan, Yaakub, & Ramli, 2021). A throttle input provides the needed climb or descent to the WIG craft to maintain its desired altitude. Thus, an altitude-based throttle controller was designed

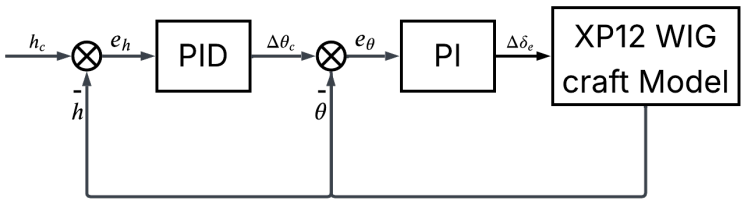


Figure 1: Pitch control loop.

to minimize the height error and output the desired throttle command. The throttle controller is a PI control loop as shown in the Fig. 2. Our WIG craft is designed such that only forward thrust is enabled. Hence, the output of the throttle control loop is constrained between 0 and 1, implying that the reverse thrust capability is currently disabled. This simplification facilitates initial testing and validation. In subsequent field trials, the reverse thrust functionality will be activated, as it will prove highly beneficial for performing docking maneuvers near the coast or in confined water environments.

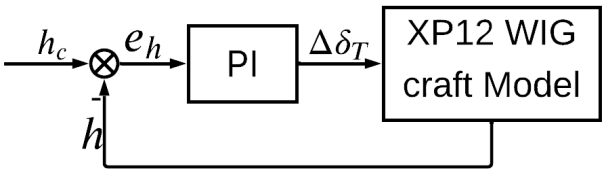


Figure 2: Throttle control loop.

An airspeed-based throttle control strategy, consistent with classical longitudinal decoupling in UAVs (Johansen & Fossen, 2020) is under consideration. By using airspeed as the error input instead of altitude, the energy management system will better mitigate the strong ground-effect-induced disturbances.

Lateral Control

Lateral Control - Heading

The lateral control of a WIG craft includes control of the heading and roll degrees of freedom. After the WIG craft had achieved longitudinal stability, a rudder-based heading controller was implemented. The control loop is displayed in Fig. 3. The task of the heading controller was to correct the yaw DOF. Whenever a waypoint is provided to the WIG craft, the controller calculates the desired heading command in order to reach the waypoint. The waypoint is represented by GPS coordinates. The desired heading calculator determines the optimal direction to steer the WIG craft to efficiently progress towards a waypoint, using both current and desired coordinates.

Heading calculator: The calculation of the compass bearing between two points on the Earth's surface, given by their latitude and longitude coordinates, relies on spherical trigonometry (Kaplan & Hegarty, 2006). Let the coordinates of the first point be (ϕ_1, λ_1) and the second point be (ϕ_2, λ_2), where ϕ denotes latitude and λ denotes longitude in radians. The objective is to compute the initial bearing θ that one must follow from the first point to reach the second point along the shortest path on the surface of the Earth (great-circle path).

The difference in longitude is given by:

$$\Delta\lambda = \lambda_2 - \lambda_1. \tag{34}$$

Using this, the bearing θ can be calculated as:

$$\begin{aligned}\theta = \arctan 2\Big(& \sin(\Delta\lambda) \cdot \cos(\phi_2), \\ & \cos(\phi_1) \cdot \sin(\phi_2) \\ & - \sin(\phi_1) \cdot \cos(\phi_2) \cdot \cos(\Delta\lambda)\Big)\end{aligned} \tag{35}$$

where $\arctan 2(y, x)$ is the two-argument arctangent function, which computes the angle whose tangent is $\frac{y}{x}$ and places it in the correct quadrant (Kai, Hamel, & Samson, 2018). The output of $\arctan 2$ ranges from $-\pi$ to π.

Heading control PID loop: After comparing it with the current yaw angle of the body axis of the WIG craft, a heading command is calculated. This heading command is then transformed into a value between -1 and 1, which is then sent to the simulator. The heading loop is a PID controller.

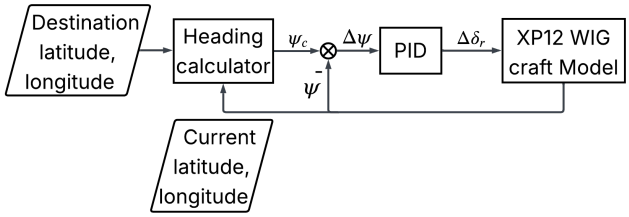


Figure 3: Heading control loop.

Lateral Control - Roll

The second component of lateral control of the WIG craft involves the roll motion. With pitch and heading control working correctly, it was observed that the heading-based control alone is sufficient for minimal waypoint tracking capability. However, the reaction was slow only when the rudder was used and altitude changes were inefficient without engaging the ailerons. Therefore, a roll controller was designed for efficient maneuvers, allowing both height adjustments and reactive banking. PID controllers based on ailerons were preferred for their swift response (Ayora, 2017). Incorporating roll-based banking will also allow the WIG craft to perform sharp turns, which is essential for obstacle avoidance.

The roll controller, similar to the altitude controller, is a cascaded control loop. A cross-track distance error, as seen in the Fig. 4, was calculated in order to keep the WIG craft in the desired path. A cross-track distance is the shortest distance of a point from a straight line. In our case, the straight line represents the shortest distance between two waypoints, with one being the source waypoint and the other being the waypoint to be reached. It guides the WIG craft to stay on the straight line, the shortest path, as it moves from one waypoint to another. Flying close to the surface limits the maximum banking angle of the WIG craft; hence, a limit of $\pm 5°$ roll angle was set.

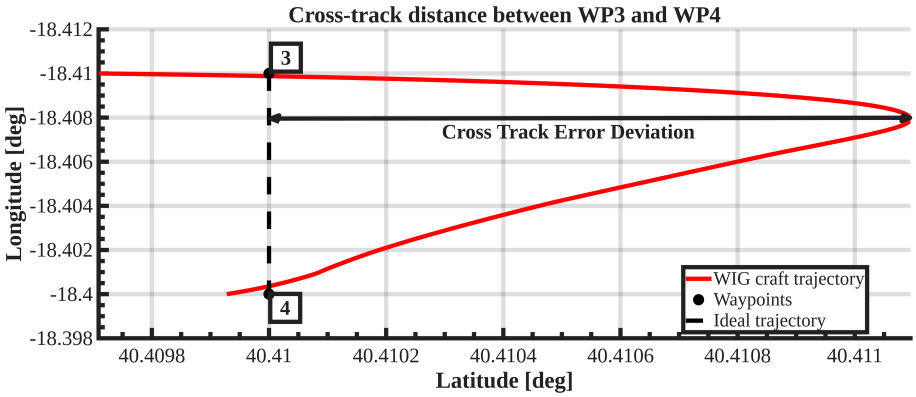


Figure 4: **Cross-Track Error (XTE) calculation**. A part of the actual path is shown in this figure. The perpendicular distance of a position of the WIG craft from the ideal straight line trajectory is known as the Cross-track distance. This error is then minimized by roll angle inputs.

Cross-Track Error Calculator: The cross-track distance between the current position of a WIG craft and the planned flight path (also known as the cross-track error, XTE) is a critical metric in guidance and navigation. It represents the lateral deviation of the WIG craft from its intended route and directly influences the control actions needed to correct the trajectory (Lekkas & Fossen, 2014; Nugroho, Dharmawan, Lelono, & Handayani, 2022). The PID controller continuously aims to minimize this error, ensuring smooth and efficient waypoint tracking.

Mathematically, the Cross-Track Error (XTE) can be computed using the spherical law of sines as follows:

$$XTE = d_{xt} = r \arcsin\left(\sin\left(\frac{d_{start,pt}}{r}\right) \cdot \sin\left(\theta_{start,pt} - \theta_{path}\right)\right) \tag{36}$$

where XTE is the lateral distance of the WIG craft from the planned path, $d_{start,pt}$ is the great-circle distance from the starting waypoint to the WIG's current position, r is the Earth's radius (assumed spherical $\approx$ 6371 km), $\theta_{start,pt}$ is the initial bearing from the starting waypoint to the WIG craft's position, θ_{path} is the bearing of the intended flight path between waypoints.

This formula effectively projects the WIG's position deviation onto the perpendicular axis relative to the desired flight path, accounting for the Earth's curvature, which is especially important for long-distance navigation (Lekkas & Fossen, 2014; Nugroho et al., 2022). The cross-track error is a foundational variable in waypoint navigation and path-following control algorithms, enabling real-time corrections in the lateral axis to maintain the WIG craft on the desired trajectory (Lekkas & Fossen, 2014).

Roll control PD-PI loop: With the cascaded PD-PI control architecture, a roll controller can be implemented to regulate the WIG craft's ailerons. The outer loop, which is a PD controller, computes the cross-track error and generates a desired roll angle. The desired roll is then compared with the current roll to calculate the roll error, which is used to produce the aileron command in the range $[-1, 1]$ for the simulator. The overall structure of this cascaded roll control system is shown in the Fig. 5.

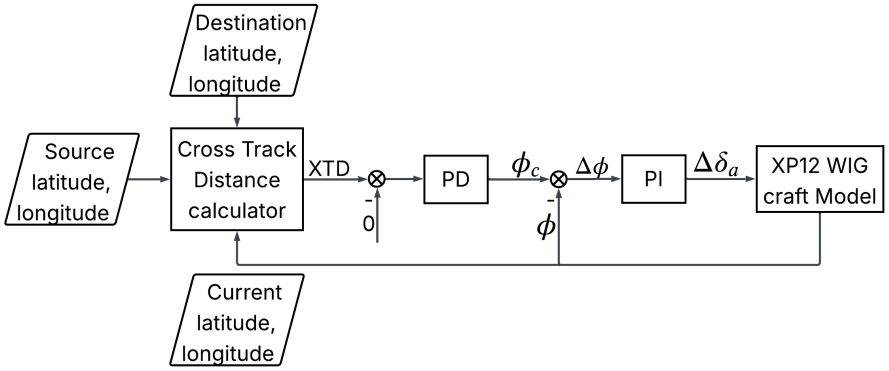


Figure 5: Roll control loop.

3.1.2 Waypoint Tracking Guidance

In this section, the design of a PID-based longitudinal and lateral control architecture of the WIG craft will be presented. The proposed control system enables the vehicle to autonomously accomplish the entire mission in simulation, from the start of the run through accurate tracking of all pre-defined way-points and culminating in a landing. The complete guidance flowchart is described in Fig. 6, while the geodesic distance calculation based on Haversine formulation is discussed in the following section.

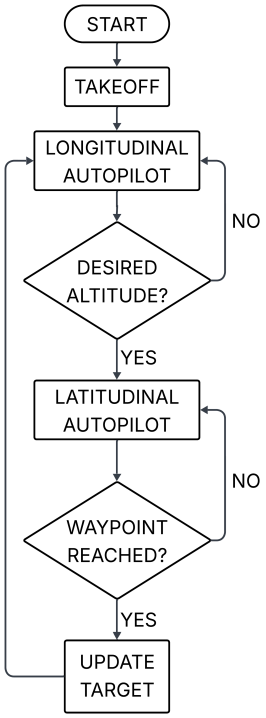


Figure 6: Complete guidance flowchart.

Haversine Formula

The shortest path between two points on a sphere is estimated as the great circle distance with the use of the Haversine formula (Robusto, 1957). It is calculated assuming that the Earth is a perfect sphere. The points are defined as GPS coordinates. To determine whether the WIG craft has reached a waypoint, it is essential to establish a threshold. This threshold is known as the great circle radius. If the WIG craft is within the great circle distance, it is assumed that the waypoint has been reached, following which, the next waypoint will be updated. For this task, the Haversine formula is used. A radius threshold of 100 m was kept. If the WIG craft is within a radius of 100 m around a waypoint, it means that the waypoint has been reached. Subsequently, the next waypoint is updated in the controller.

To calculate the great circle distance, the Haversine formula utilizes an intermediate value, a, representing the squared half-chord length between the points, which accounts for the latitude and longitude differences. The great-circle distance d is then computed by applying the inverse sine function to the square root of a, scaled by the Earth's radius.

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{37}$$

$$d = 2r \arcsin\left(\sqrt{a}\right) \tag{38}$$

$$\Delta\varphi = \varphi_2 - \varphi_1 \tag{39}$$

$$\Delta\lambda = \lambda_2 - \lambda_1 \tag{40}$$

where φ_1, φ_2 are the latitudes of the two points (in radians), λ_1, λ_2 are the longitudes of the two points (in radians). r is the radius of Earth ($\approx$ 6371 km) and d is the great-circle distance between the two given points.

3.1.3 Experimental Results

This section presents simulation outputs from tests conducted using the PID-based controller, demonstrating the effectiveness of the proposed control architecture for autonomous navigation of WIG vehicles. The simulations were conducted using the FAA-certified X-Plane 12 flight simulator, which was initially integrated and configured within WP3 by Tampere University. All experiments were conducted on a PC running Linux OS with an Intel Core i9 processor with 32 GB RAM, with the control loops operating at 50 Hz.

The controller architecture described previously was evaluated through multiple waypoint following missions designed to maintain the WIG craft in ground effect at a target altitude of 1.0 m above the water surface. All results correspond to identical initial conditions and a wind-free baseline condition.

2D "square" path

In this first experiment, four waypoints were arranged in a square configuration, and the WIG craft was tasked to navigate through each of them sequentially, traversing a square trajectory. The 2D mission path is shown in Fig. 7. The axes represent the latitude and longitude of the waypoints. The mission starts at GPS coordinate $(40.40°, -18.40°)$, a region of the North Atlantic Ocean. The WIG craft followed the square path with stable behavior and maintained a small cross-track error while traversing each waypoint.

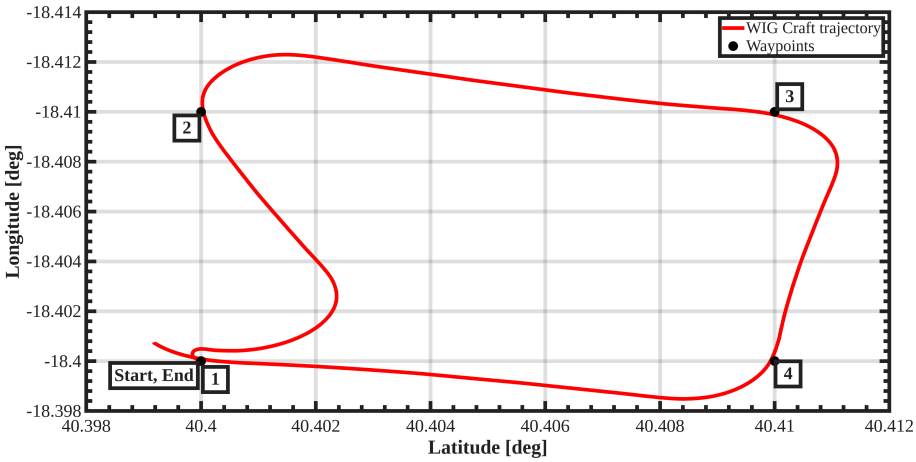


Figure 7: 2D ”square” flight path.

The altitude control maintains the WIG craft near the 1.0 m target altitude in a square path with high precision, with an RMSE of 0.321 m. The maximum altitude error of 1.372 m is consistent with small overshoot pertaining to initial take-off from water. Pitch control achieves stable output with RMSE $0.028°$ and a maximum error of $0.081°$, ensuring smooth longitudinal dynamics.

Roll tracking shows excellent performance with RMSE of $0.223°$ and maximum error constrained near $1.0°$, reflecting aggressive but safe lateral corrections. The roll controller exhibits rapid response, with mean absolute error (MAE) of $0.153°$ and maximum error of $1.010°$. Yaw control tracks heading with larger variability (RMSE $3.856°$, max error $14.059°$) reflecting the complexity of heading transitions around waypoints, with MAE of $1.211°$. Significant yaw overshoot occurs due to transient turns but remains controllable.

2D ”eight” path

The second experiment expanded the complexity by defining eight waypoints forming a figure-eight trajectory. The mission again started at $(40.40°, -18.40°)$. Using the same guidance and control architecture, the WIG craft successfully tracked each of eight waypoints, producing a smooth figure-eight path with minimal deviation from the ideal trajectory.

The altitude control shown in Fig. 9, maintains the WIG craft consistently close to the target 1.0 m altitude with small deviations in a ”eight” shape path. Despite continuous curvature changes inherent in the ”eight” trajectory, the controller achieves an RMSE of 0.072 m with maximum error of 1.361 m. The MAE is calculated to be 0.145 m compared to the square path, reflecting the increased complexity of altitude regulation during continuous turning maneuvers. The pitch control output tracks commands smoothly with RMSE of $0.031°$ and max error of $0.019°$, ensuring stable longitudinal dynamics even during continuous turns.

Roll control response successfully tracks commanded roll angles while enforcing the safety limit of $\pm 5.0°$. The roll controller demonstrates improved performance with RMSE of $0.170°$ and exceptionally fast response times (MAE $0.123°$). Due to the continuous turning nature of the ”eight” path, roll commands are more frequent but remain well controlled with maximum error limited to $2.733°$. The yaw controller effectively manages the heading transitions required by the complex trajectory, achieving RMSE of $3.231°$ with MAE of $1.772°$ and max error of $13.248°$, demonstrating bounded transient

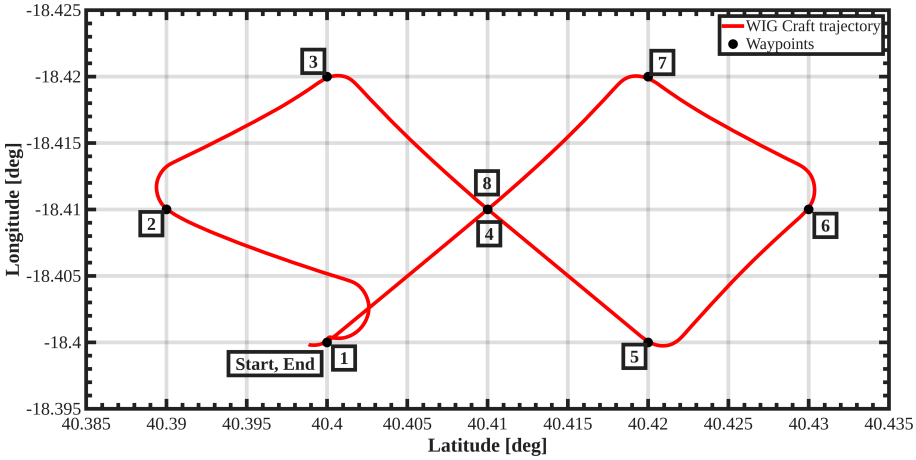


Figure 8: 2D ”eight” flight path.

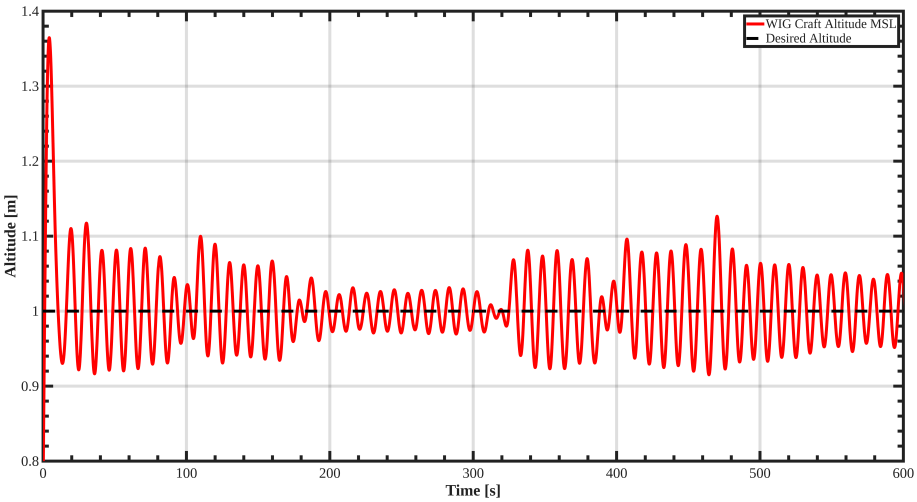


Figure 9: 2D ”eight” altitude control output.

excursions that converge within acceptable tolerances.

3D low altitude path

The third experiment introduced an additional level of difficulty by extending the waypoint-tracking task from 2D to 3D. Each waypoint was assigned an altitude, requiring simultaneous control of horizontal position and height. The waypoints were arranged to form a tilted closed ”W” pattern when viewed from above (Fig. 10), resulting in a fully 3D mission profile as illustrated in Fig. 11. The WIG craft was able to follow the 3D trajectory and manage altitude transitions effectively.

The altitude controller maintains tracking with RMSE of $0.219\,\mathrm{m}$ and maximum error of $0.844\,\mathrm{m}$, indicating increased variability due to the three-dimensional trajectory complexity. The rise time of $0.081\,\mathrm{s}$ remains reasonable, but settling time extends to $1.313\,\mathrm{s}$, reflecting the challenging nature of altitude regulation during complex 3D maneuvers. Pitch control shows increased error metrics (RMSE $0.652°$, maximum error $4.973°$) with rise time of $0.286\,\mathrm{s}$, indicating more dynamic longitudinal responses.

The 3D roll control response displayed in Fig. 12c shows effective tracking of commanded roll angles

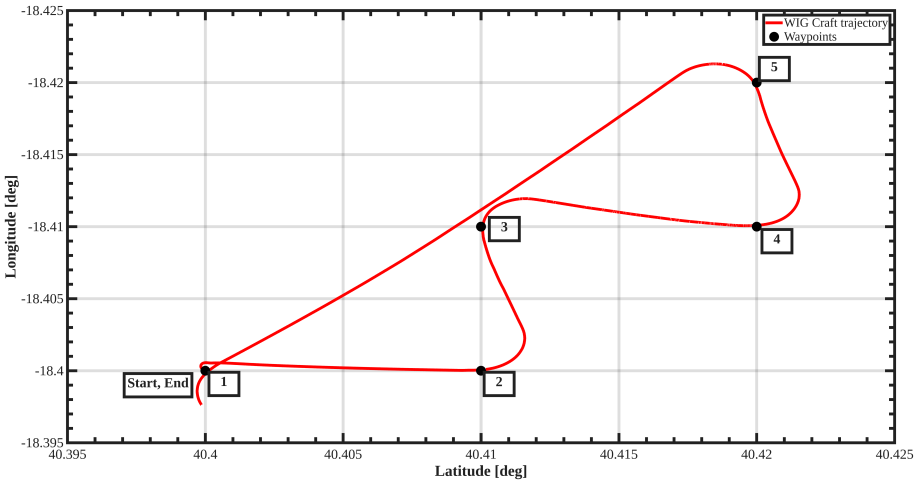


Figure 10: 3D flight path (Top view).

throughout the mission, achieving RMSE of $0.271°$ with maximum error of $1.825°$. The rise time decreases to 0.096 s and settling time to 0.015 s, reflecting greater control effort during complex turns while maintaining the $\pm 5.0°$ safety limit to prevent excessive bank angles that could compromise ground effect stability. The yaw controller performance (Fig. 12d) demonstrates heading tracking with RMSE of $3.356°$ and maximum error of $16.438°$, with rise time of 13.153 s and settling time of 0.076 s. These lateral control outputs verify the robustness of the cascaded control architecture when managing coupled lateral and longitudinal dynamics characteristic of WIG vehicle flight near the water surface.

3.1.4 Time-Domain Performance Metrics

To accurately characterize the transient response behavior of the control system, several time-domain performance metrics were computed. These metrics enable a systematic analysis of rise time, settling time, overshoot, peak time, and steady state error. This provides a quantitative insight into the dynamic performance of the control architecture and enables comparison of various controllers under identical conditions.

Rise Time (t_r)

The rise time is the time taken for the step response to rise from a lower percentage ($10\,\%$) to a higher percentage ($90\,\%$) of its final steady-state value for the first time. It quantifies how quickly the system responds initially to a step input.

$$t_r = t_{90} - t_{10}$$

where

$$y(t_{10}) = 0.1\, y_{\text{final}}, \qquad y(t_{90}) = 0.9\, y_{\text{final}}$$

and y_{final} is the final steady state value of the step response.

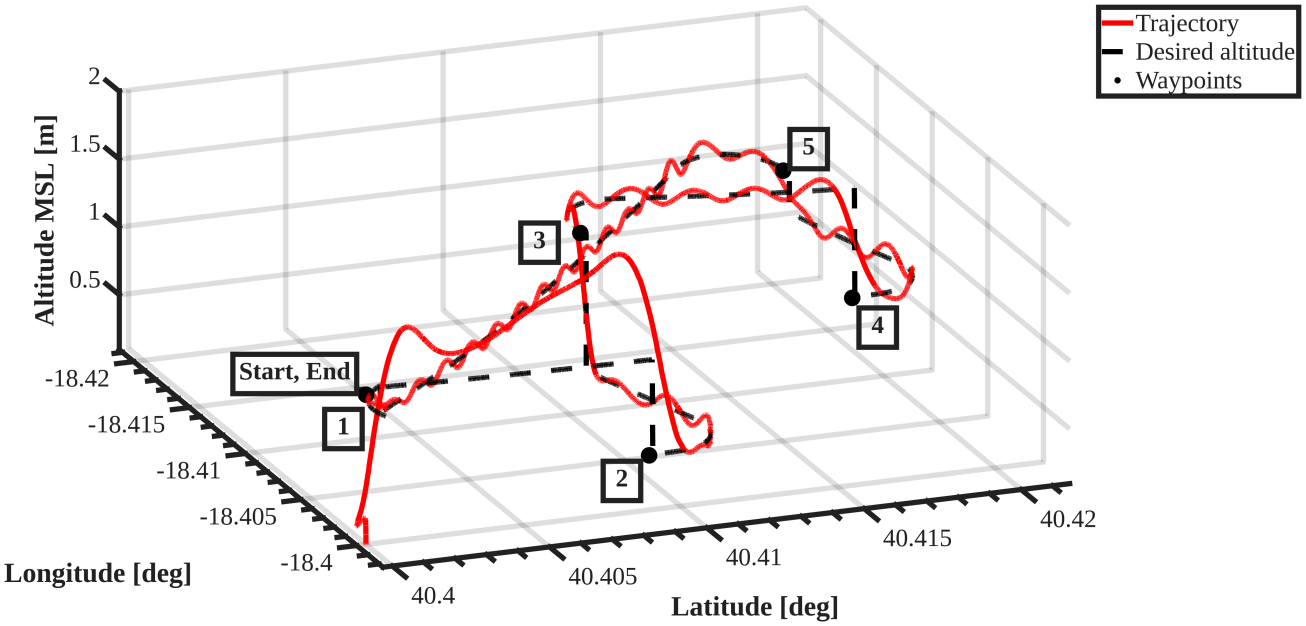


Figure 11: 3D flight path. The axes represent latitude and longitude of the waypoints. The mission starts at GPS coordinate $(40.40°, -18.40°)$, a region of the North Atlantic Ocean. The WIG craft is tasked to simultaneously control position and height.

Settling Time (t_s)

The settling time is the time required for the step response to remain within a percentage band (commonly $\pm 2\,\%$ or $\pm 5\,\%$) around the steady-state value, indicating when the transient effects have effectively died out.

$$t_s = \min\{t : |y(t) - y_{\text{final}}| \leq \delta \times y_{\text{final}}; \forall t \geq t_s\}$$

where δ is the settling tolerance band (e.g., 0.02 for $2\,\%$).

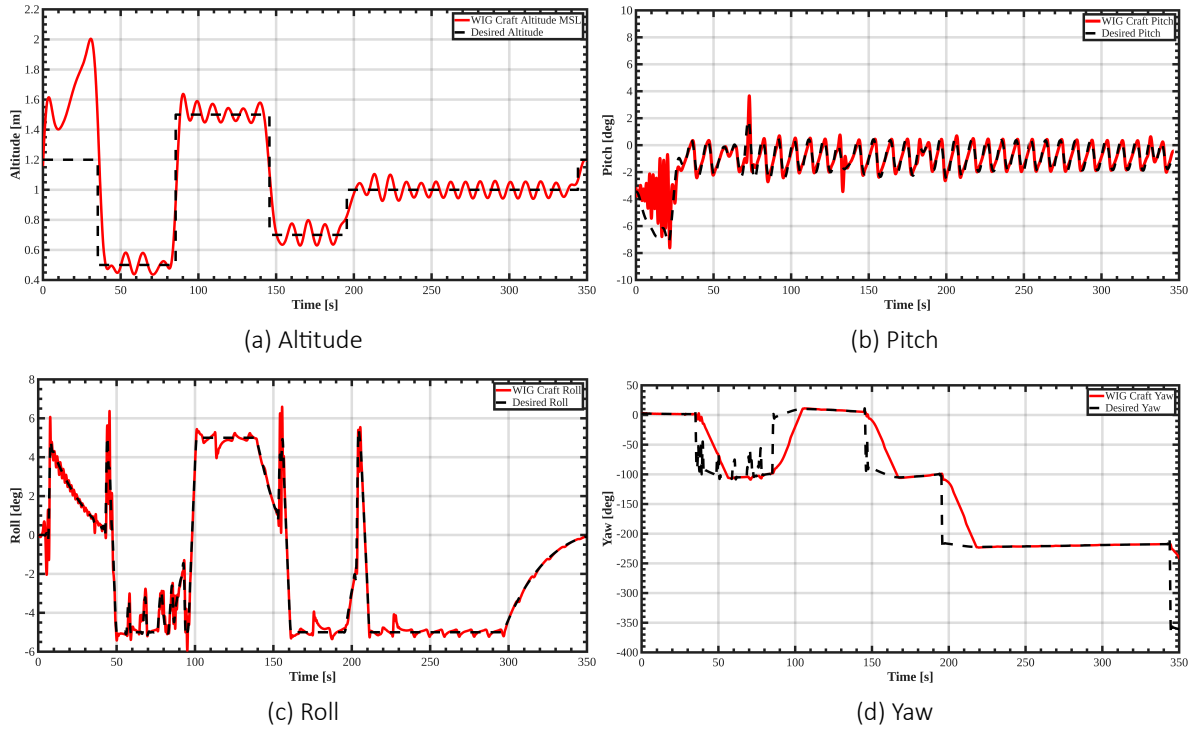
Overshoot (M_p)

Overshoot measures how much the peak response exceeds the steady-state value, expressed in absolute units of the final value. It indicates the extent of transient exceeding or oscillation before settling.

$$M_p = \max_t (y(t) - y_{\text{final}})$$

Often expressed as a percentage overshoot:

$$\%M_p = \frac{M_p}{|y_{\text{final}}|} \times 100\%$$



(a) Altitude (b) Pitch

(c) Roll (d) Yaw

Figure 12: 3D control outputs for altitude, pitch, roll, and yaw.

Peak Time (t_p)

Peak time is the time at which the maximum overshoot (peak response) occurs. It is indicative of the rapidity of maximum transient excursion.

$$t_p = \arg\max_t(y(t))$$

Steady State Error (e_{ss})

The steady state error is defined as the absolute difference between the step input and the desired step response value once the system has reached its equilibrium, that is when all transient dynamics have vanished. This metric is key to deciding whether the controller needs a persistent corrective input to maintain the reference trajectory.

$$e_{ss} = \lim_{t\to\infty} |y(t) - y_{\text{final}}|$$

3.1.5 Tracking Performance Metrics

To accurately evaluate the tracking and control performance, several error metrics were computed between the measured values y_i and the reference values $\hat{y}_i$, over N samples. The proposed cascaded control architecture achieves robust altitude and heading regulation across all mission profiles.

Root Mean Square Error (RMSE)

The RMSE measures the square root of the average of the squared differences between observed and reference values. It captures the magnitude of the typical prediction error, giving relatively higher weight to large errors, making it sensitive to outliers. RMSE is commonly used because it has the same units as the measured variable, facilitating interpretation of the error scale.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

Mean Absolute Error (MAE)

The MAE computes the average of the absolute differences between observed and reference values, reflecting the average magnitude of the errors regardless of their direction. This metric is less sensitive to large deviations than RMSE, providing a complementary measure of typical error magnitude.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Maximum Error (MaxErr)

MaxErr identifies the single largest absolute deviation between measured and expected data points. This worst-case error measurement is useful to assess robustness and detect critical failures or outliers in the system performance.

$$\text{MaxErr} = \max_{1\leq i\leq N}|y_i - \hat{y}_i|$$

Table 5: Time-domain performance metrics

States	t_r [s]	t_s [s]	M_p [%]	t_p [s]	e_{ss}
Altitude [m]	0.081	1.313	11.91	30.283	0.334
Pitch [deg]	0.286	0.016	4.02	71.646	0.025
Roll [deg]	0.096	0.015	6.45	45.486	0.072
Yaw [deg]	13.153	0.076	1.13	60.207	0.098

The quantitative estimates (tracking performance metrics and time-based performance metrics) of the waypoint tracking task have been provided in the Table 5 and Table 6, respectively.

Overall, the proposed controller provides accurate altitude and heading tracking while respecting safety limits at the same time. The 3D trajectory (Fig. 11) reveals the coupling between lateral maneuvers and vertical control, motivating either a coordinated-turn term in the altitude controller or a feed-forward compensation from the lateral controller in future works. The actuator saturation fraction is low, indicating the commands are within the system's capability for the nominal mission.

Table 6: Tracking performance metrics

State	Metric	2D Square	2D-8	3D
Altitude	RMSE [m]	0.321	0.072	0.219
	MAE [m]	1.143	0.145	0.924
	MaxErr [m]	1.372	1.361	0.844
Pitch	RMSE [deg]	0.028	0.031	0.652
	MAE [deg]	0.012	0.022	0.906
	MaxErr [deg]	0.081	0.019	4.973
Roll	RMSE [deg]	0.223	0.170	0.271
	MAE [deg]	0.153	0.123	0.186
	MaxErr [deg]	1.010	2.733	1.825
Yaw	RMSE [deg]	3.856	3.231	3.536
	MAE [deg]	1.211	1.772	2.727
	MaxErr [deg]	14.059	13.248	16.348

3.2 Linear Quadratic Gaussian (LQG) Control

In this section, a Linear Quadratic Gaussian (LQG) approach for controlling an unmanned WIG vehicle is presented. This work has been published at IEEE 2024 7th Iberian Robotics Conference (ROBOT) (Ilić, Milošević, Suarez Fernandez, Rossi, & Domínguez, 2024).

A nonlinear and non-stationary state-space 6DoF model is obtained by system identification, around the trimmed operating point. Implementing the proposed control strategy includes defining the desired trajectory of motion, linearizing the model around the trim operating point, and computing adequate controller gains. The feasibility of implementing the proposed approach has been demonstrated through system behavior simulations in various conditions using Matlab and Simulink.

LQG strategy is an approach in control theory that belongs to linear control techniques that optimize quadratic functions. It combines two well-known and most-used concepts in control theory: the LQ approach and the Kalman filter. The LQ concept aims to find the optimal control law that minimizes the quadratic cost function, considering system states and control signals. The implementation of this strategy assumes knowledge of system dynamics and noise-free measurements. The Kalman filter estimates the system's state by combining measurements with noise and knowledge of the system's dynamics. It provides an optimal state estimation, even in the presence of noise in the measurements. Therefore, the LQG controller ensures a stable and robust system response to disturbances and noise, and it is widely used for controlling Multiple Input and Multiple Output (MIMO) systems.

Before applying the LQG control strategy technique, it is necessary to select operating points and linearize the model around them. Then, based on the linearized model, the LQG controller is designed. The LQG control technique aims to minimize the following criterion function

$$J = \int_0^\infty (x(t)^T Q x(t) + u(t)^T R u(t))\, dt \tag{41}$$

by generating the control signal

$$u(t) = -K(t)x(t) \tag{42}$$

where Q and R are weighting matrices, $K(t)$ gain which is calculated from the following equation

$$K(t) = -R(t)^{-1} B(t)^T S(t) \tag{43}$$

in which $S(t)$ is the solution to the Riccati differential equation.

The first step in designing an LQG controller is to calculate the LQ optimal gain. For the calculation of the optimal gain, it is necessary to adopt appropriate weighting matrices that significantly affect the controller performance. There are certain empirical recommendations for adopting the initial values of the weight matrices. Then, the optimal values are determined by observing the system's behavior for the adopted parameters. For the observed system, the weight matrices were determined using the trial-and-error method, starting from the recommended values until matrix values that achieved satisfactory results were found. Based on the weight matrices, by solving the Riccati equation, the corresponding LQ gains were calculated, and then the Kalman filter was designed for the estimation of all variable states. It is important to note that as input variables of the Kalman filter, correction control signals and obtained deviations of the output variables from the nominal values are passed on, and then the estimated deviations of the state variables are multiplied by the LQ gain and added to the nominal control signal. The control system with an incorporated LQG controller that generates the corrective control signal is presented in the following block scheme in Fig. 13, where the plant block represents the behavior of the system when the trim control values are applied.

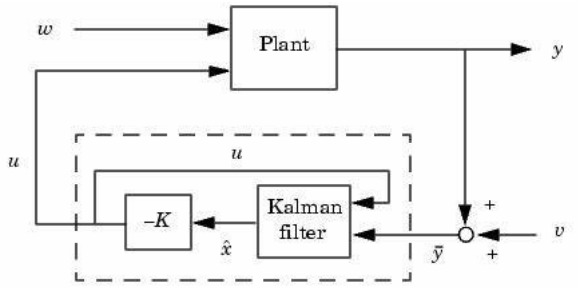


Figure 13: LQG Block Scheme

3.2.1 Trimming the WIG Vehicle

Adopting the operating working point is a necessary step for linearizing the nonlinear equations describing the system dynamics and the subsequent application of the LQG controller. For aerial navigation, the operating points are usually chosen as longitudinal equilibrium states in which the velocity and gravity vectors lie in the plane of symmetry of the vehicle, i.e., operating points that correspond to flight modes such as steady flight, climbing, or descending. Since the principle of flying a WIG vehicle is based on the maximum use of the ground effect in steady flight, it is desirable to ensure a steady flight at a certain speed without a change in direction. In steady flight, the WIG vehicle balances out forces in two directions. Vertically, the lift force matches the gravity force so the vehicle neither rises nor falls. Horizontally, the thrust force equals the drag force so the vehicle moves at a constant speed. This balance ensures a smooth and stable flight. Also, in steady flight, there is no change in the starting values of the state variables, therefore it is valid that the first derivatives of the nine state variables are equal to zero. The objective was to compute trim states and inputs when the WIG vehicle is in cruise steady-state flight, simultaneously satisfying the following conditions:

- Travelling at constant trimmed steady-state flight speed $V_a^* = 85m/s$, corresponding to the cruise speed of the AIRSHIP Commercial Concept (CC) with a 25 m wingspan.
- Constant flight path angle γ equal to zero

- Constant speed $\dot{u}^* = \dot{v}^* = \dot{w}^* = 0$
- Constant roll and pitch angles $\dot{\phi}^* = \dot{\theta}^* = \dot{p}^* = \dot{q}^* = 0$

The control and state values, which ensure the presented trim condition, were calculated using numerical optimization, considering the limitations of the control signals. By minimizing the criterion function $J_2 = P^T W P$, where P is a vector of previously introduced trim conditions and W is the weighted identity matrix, the following trim values of control signals, states, and their first derivatives for applied trim control values were obtained using the Nelder-Mead simplex search method (Lagarias, Reeds, Wright, & Wright, 1998)

$$u^* = [0, -0.178, 0, 0.0821, 0.0821]^T \tag{44}$$

$$x^* = [84.9905, 0, 1.2713, 0, 0, 0, 0, 0.0150, 0]^T \tag{45}$$

$$\dot{x}^* = [0]_{9x1} \tag{46}$$

3.2.2 Linearization around operating trim point

Model linearization around the previously chosen operating trim point was performed using the small perturbation theory. Because the functions in the state space system model around the chosen operating point are time-invariant, they can be represented with first-order Taylor expansion. The resulting linearized model is given in matrix form as

$$\frac{\mathrm{d}\Delta x(t)}{\mathrm{d}t} = A\Delta x(t) + B\Delta u^*(t) \tag{47}$$

$$\Delta y(t) = H\Delta x(t) + D\Delta u^*(t) \tag{48}$$

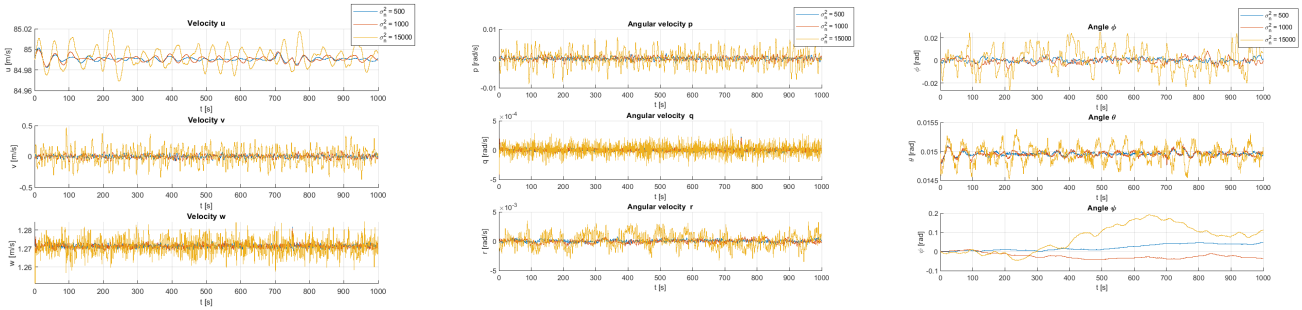
where $H = I_{9x9}$, $D = [0]_{9x9}$, $\Delta u^*(t) = \begin{bmatrix} \Delta u(t) \\ 1 \end{bmatrix}$.

As a final step, the controllability, observability, and stability of the linearized system were analyzed using Kalman's tests and by plotting the pole-zero graph. The linearized system was found to be controllable and observable, with marginal open-loop stability, which is consistent with the known inherent instability of WIG vehicle dynamics near the ground-effect operating regime.

3.2.3 Simulation results

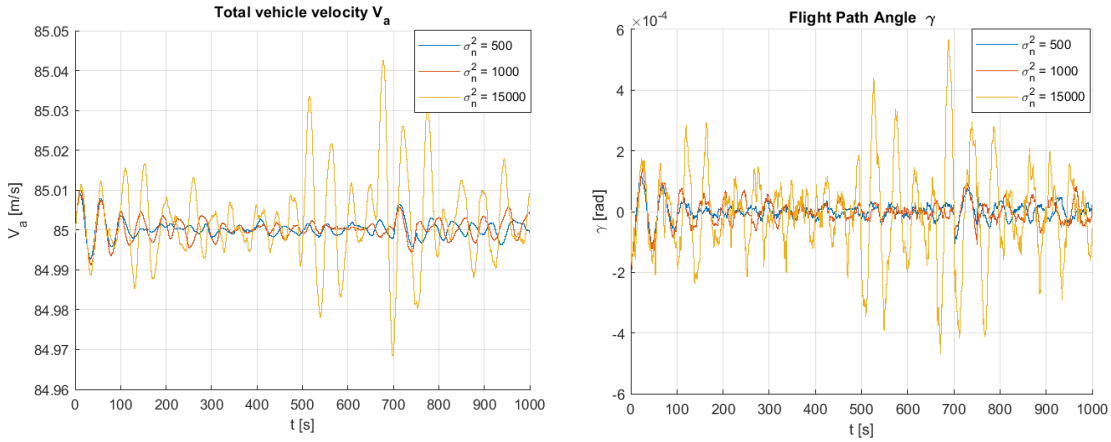
After the model and controller were implemented in the chosen software Matlab and Simulink, measurement noise and disturbance were modeled and used for robustness analysis of system behavior. The behavior of the system was simulated for 1000 seconds with different Gaussian white noise present in the measured state signals and disturbance applied to the state u during the time interval $[700, 750]s$ with amplitude 50% of the trim value of the state. The simulation results of the closed-loop system with the LQG controller and the Kalman filter with the conditions presented are shown in the following Fig. 14a to Fig. 16b. Furthermore, to analyze compliance with the conditions previously introduced for steady-state flight, Table 7 shows squared errors for the most important variables.

From the graphs shown, it can be noticed that as the noise level increases, deviations from the nominal values and oscillations follow the same trend and are visible in all state variables. Nevertheless, these deviations are very small-scale, and the controller manages to cope with this disturbance. Looking at the behavior during the interval of action of the disturbance in the state variable u, greater peaks present in the total velocity V_a can be seen, especially at the moment of activating and deactivating the



(a) Linear velocities (b) Angular velocities (c) Euler angles

Figure 14: LQG control results for velocity and attitude-related states.



(a) Total vehicle velocity V_a (b) Flight path angle γ

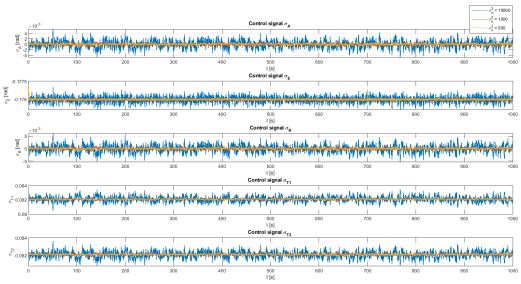
Figure 15: LQG control results for velocity and flight path angle.

disturbance. Furthermore, the maximum amplitude of the total velocity increases with the increase of the noise level; still, the deviations from the nominal signal are insignificant, in the value range of approximately 10^{-5} to 10^{-3}, with respect to the increase in the noise level, as shown in Table 7.

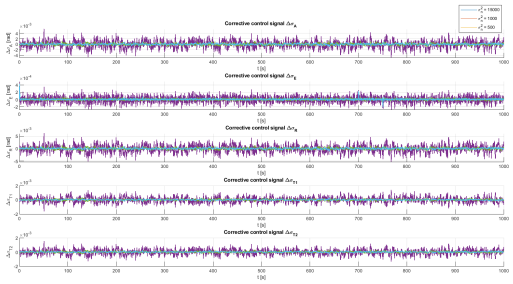
Table 7: Mean squared error (MSE) for different noise levels

σ_n^2	500	1000	15000
MSE_{V_a}	9.9001×10^{-5}	1.061×10^{-4}	1.6868×10^{-4}
MSE_{γ}	2.2369×10^{-4}	2.2370×10^{-4}	2.2355×10^{-4}
$MSE_{\dot{u}}$	9.4410×10^{-8}	1.4852×10^{-7}	2.3048×10^{-6}
$MSE_{\dot{v}}$	2.7449×10^{-4}	5.1591×10^{-4}	7.4713×10^{-3}
$MSE_{\dot{w}}$	2.6538×10^{-6}	4.1264×10^{-6}	4.0323×10^{-5}
$MSE_{\dot{\phi}}$	2.3448×10^{-7}	4.5708×10^{-7}	7.3684×10^{-6}
$MSE_{\dot{\theta}}$	4.4878×10^{-10}	6.9466×10^{-10}	6.7844×10^{-9}
$MSE_{\dot{p}}$	1.0328×10^{-7}	1.9391×10^{-7}	2.8611×10^{-6}
$MSE_{\dot{q}}$	2.6649×10^{-9}	4.4928×10^{-9}	5.0282×10^{-8}

Altogether, increasing noise levels leads to consistent oscillations across all state variables, although these remain minor, and the controller effectively mitigates them. During disturbance presence in the



(a) Control signals



(b) Corrective control signals

Figure 16: LQG control results.

state variable u, notable peaks occur in total velocity V_a, with peak amplitudes increasing alongside noise levels, yet the controller preserves insignificant deviations from the nominal signal.

The presented results demonstrate that the LQG controller can robustly regulate the WIG vehicle dynamics around a steady-state ground-effect operating point in the presence of measurement noise and external disturbances. The simulations confirm the feasibility of applying linear optimal control techniques to the AIRSHIP WIG vehicle for steady-flight conditions.

3.3 Linear Quadratic Regulator (LQR) Control

The Linear Quadratic Regulator (LQR) is type of optimal state-feedback control law that minimizes a quadratic cost function $\boldsymbol{J}$ for a linear time-invariant (LTI) system of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t), \tag{49}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t), \tag{50}$$

where $\boldsymbol{x} = [u\ v\ w\ p\ q\ r\ q_0\ q_1\ q_2\ q_3\ x\ y\ z]^T$ is our system state space vector and $\boldsymbol{u} = [\delta_{t_1}\ \delta_{t_2}\ \delta_a\ \delta_e\ \delta_r]^T$ is the control input. Given that the aircraft kinematics (eqs. 1 and 2) and dynamics (eqs. 5 and 6) described in Section 2 are non-linear, means that, in order to properly design a suitable LQR for our system, the plant must be linearized about trimmed flight conditions, which in this case were chosen to be $\boldsymbol{x}_{trim} = [28\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ -1]^T$. The linearization was performed using *MATLAB*'s linearization tool to directly compute the state-space matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$.

The LQR objective is to regulate the states to the origin while balancing tracking performance and control effort. This objective is formalized through a quadratic cost functional

$$\boldsymbol{J} = \int_0^\infty \left(\boldsymbol{x}^T(t)\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}(t)^T\boldsymbol{R}\boldsymbol{u}(t) + 2\boldsymbol{x}^T(t)\boldsymbol{N}\boldsymbol{u}(t)\right) dt, \tag{51}$$

where $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{N}$ are designer-selected weighting matrices. The matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ penalize state deviations and control effort, respectively, where $\boldsymbol{N}$ penalize cross products of $\boldsymbol{x}$ and $\boldsymbol{u}$.

Minimization of this functional yields the stabilizing state feedback (Kwakernaak & Sivan, 1972):

$$\boldsymbol{u}(t) = -\boldsymbol{K}\boldsymbol{x}(t), \tag{52}$$

where

$$\boldsymbol{K} = \boldsymbol{R}^{-1}\left(\boldsymbol{B}^T\boldsymbol{P} + \boldsymbol{N}^T\right), \tag{53}$$

and $\boldsymbol{P}$ is the unique positive-definite solution of the continuous-time algebraic Riccati equation

$$\boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} - (\boldsymbol{P}\boldsymbol{B} + \boldsymbol{N})\,\boldsymbol{R}^{-1}\left(\boldsymbol{B}^T\boldsymbol{P} + \boldsymbol{N}^T\right) + \boldsymbol{Q} = 0. \tag{54}$$

The LQR solution guarantees closed-loop stability when the pair $(\boldsymbol{A}, \boldsymbol{B})$ is stabilizable, meaning that, following the *Hautus Lemma for stabilizability* (Sontag, 1998) , it must satisfy

$$rank\left([\boldsymbol{\lambda}\boldsymbol{I}_n - \boldsymbol{A}\ \boldsymbol{B}]\right) = n, \text{ for all } \lambda \in \mathbb{C}^+ \tag{55}$$

where $\boldsymbol{\lambda}$ are the eigenvalues of $\boldsymbol{A}$ and n is the number of states of the system.

LQR also provides favourable robustness margins to moderate plant variations due to the convex quadratic structure of the cost function. These properties make LQR an effective baseline controller for regulating and rejecting disturbances within the linear operating envelope.

3.3.1 Linear Quadratic Regulator with Integral action

Since the standard LQR formulation regulates deviations around the origin, this leads to steady-state offsets when tracking constant references or when subjected to model uncertainties and persistent disturbances. To eliminate these offsets, integral feedback can be embedded within the optimal control framework to drive the steady-state tracking error to zero by extending the system state vector (Kwakernaak & Sivan, 1972).

The tracking error is defined as

$$\boldsymbol{e}(t) = \boldsymbol{r}(t) - \boldsymbol{C}\boldsymbol{x}(t), \tag{56}$$

and its integral

$$\boldsymbol{x}_I(t) = \int_0^t e(\tau)d\tau. \tag{57}$$

The augmented state vector becomes

$$\tilde{\boldsymbol{x}}(t) = \begin{bmatrix}\boldsymbol{x}(t)\\ \boldsymbol{x}_I(t)\end{bmatrix}, \quad \tilde{\boldsymbol{A}} = \begin{bmatrix}\boldsymbol{A} & 0\\ -\boldsymbol{C} & 0\end{bmatrix}, \quad \tilde{\boldsymbol{B}} = \begin{bmatrix}\boldsymbol{B}\\ \boldsymbol{0}\end{bmatrix}. \tag{58}$$

Applying the LQR formulation to this augmented system yields the feedback policy

$$\boldsymbol{u}(t) = -\boldsymbol{K}\boldsymbol{x}(t) - \boldsymbol{K}_I\boldsymbol{x}_I(t) \tag{59}$$

where

$$\boldsymbol{K}_{aug} = [\boldsymbol{K}\ \boldsymbol{K}_I]^T, \tag{60}$$

is obtained from the Riccati solution of the augmented system. The LQRI controller, therefore, regulates the original states while eliminating steady-state tracking error for constant references and disturbances through integral feedback. A block diagram can be seen in figure 17. Figure 18 shows the effect of adding integral action to eliminate steady-state tracking errors.

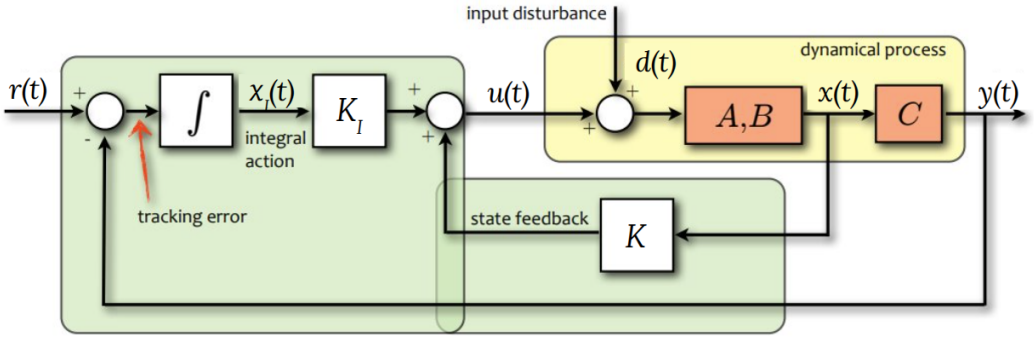
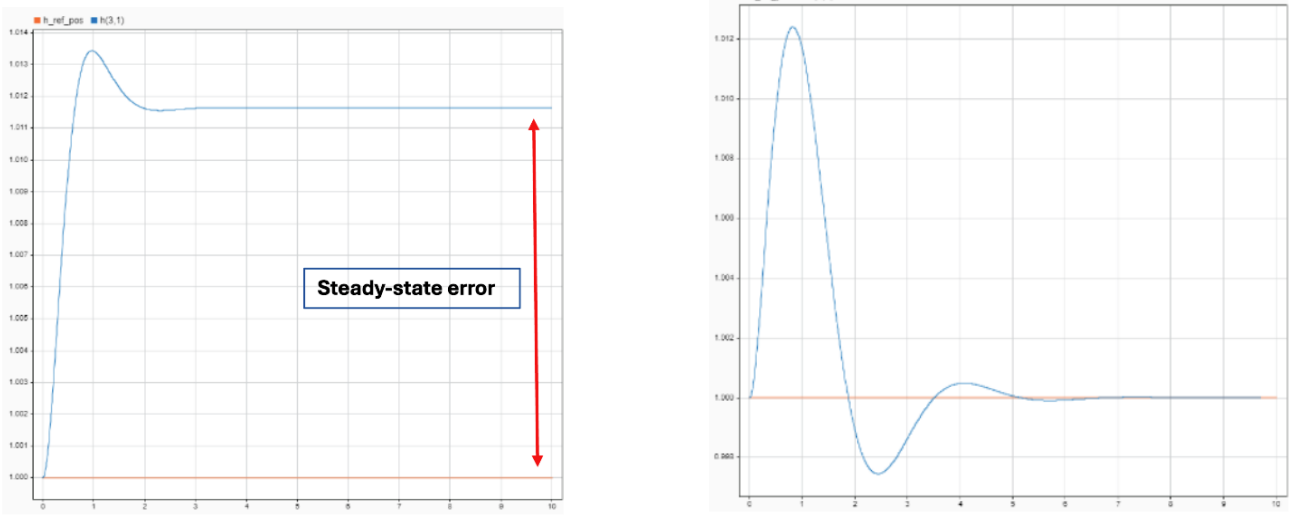


Figure 17: LQRI controller block diagram.



(a) Reference tracking with LQR

(b) Reference tracking with LQRI

Figure 18: Comparison of tracking performance for LQR and LQRI control laws. The LQR response exhibits a non-zero steady-state error, while LQRI achieves zero steady-state error due to integral action.

The initial selection of the weighting matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ was performed using the Bryson method, which normalizes the diagonal entries based on the maximum admissible state deviations and actuator limits. This provides a systematic starting point ensuring comparable penalty scales across heterogeneous

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

Table 9: Performance metrics (RMSE, settling time, σ, overshoot, mean error, and energy utilization) for the AIRSHIP vehicle in tasks (a)–(f) with the LQRI controller.

Task	RMSE (m)	Settling Time (s)	Overshoot (m)	Error (mean $\pm$ 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

states. After this normalization, the weights were manually refined through iterative simulation to balance transient performance, steady-state accuracy, and actuator utilization.

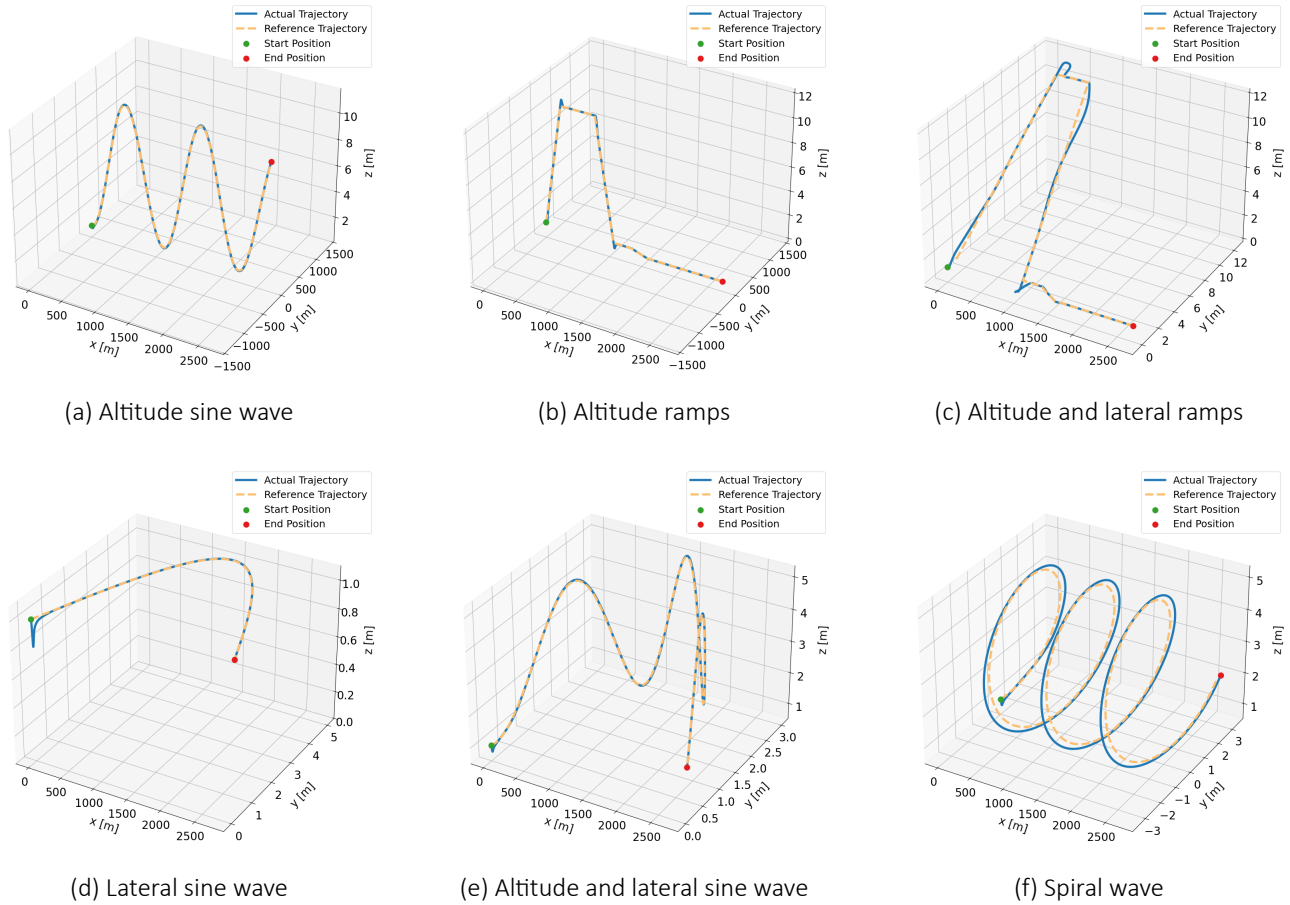
3.3.2 LQRI simulation results

To test the LQRI controller performance, six trajectory tracking tasks (with a duration of $100s$) of increasing complexity using the same AIRSHIP model were developed, and various performance metrics were used to quantitatively evaluate each task.

Table 8 summarizes these performance metrics used to evaluate control strategies. 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. Energy consumption is evaluated from integrated thrust commands (range $[0, 1]$): lower values indicate more efficient and efficient control. Conversely, excessively high energy consumption may reflect oscillatory, inefficient or unstable control behaviour.

As shown in Figure 19 and Table 9, the LQRI controller maintains low RMSE and smooth behaviour 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.

Some of these scenarios are not intended to represent nominal flight paths, rather, they are designed



(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

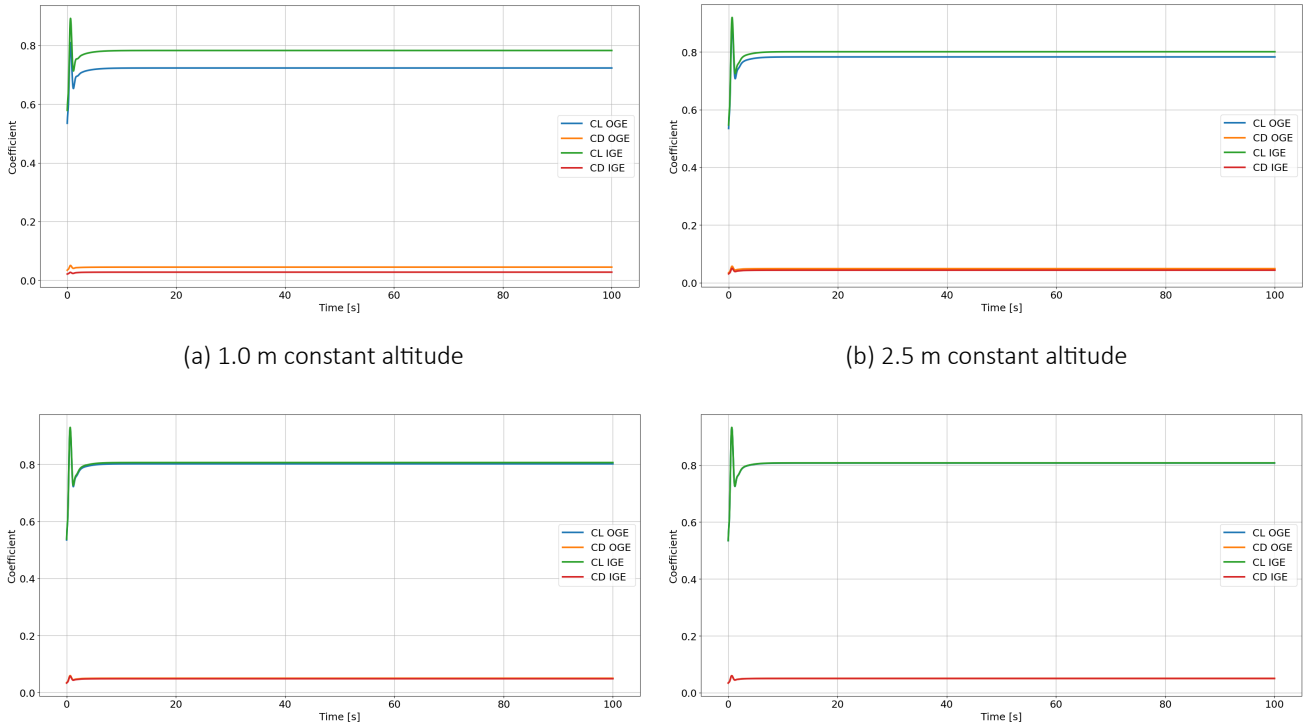
Figure 19: LQRI-controlled AIRSHIP response to tracking different trajectories.

to stress test the controllers under aggressive and unrealistic operating conditions, exposing potential weakness and ensuring robustness during the real expected flight envelope.

Table 10 presents the influence of ground effect on the performance of the LQRI 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 $(h/b) \geq 0.5$ (Fig. 20).

Table 10: LQRI performance metrics (RMSE, settling time, overshoot, error mean + std, and energy utilization) for the AIRSHIP vehicle at various constant 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) 1.0 m constant altitude

(b) 2.5 m constant altitude

(c) 5.0 m constant altitude

(d) 10.0 m constant altitude

Figure 20: Variation of C_L and C_D with altitude due to ground effect, utilizing LQRI for constant altitude tracking.

3.4 $\mathcal{L}_1$ Adaptive Control

Aircraft dynamics are inherently affected by modeling errors and parametric uncertainties, arising from variations in aerodynamic coefficients, mass properties, environmental conditions, and unmodeled dynamics. A commonly adopted industrial approach to address these uncertainties is gain scheduling; however, this method is time-consuming and requires extensive tuning, because a family of controllers must be designed and tuned for all possible operating conditions across the entire operational range (Rugh & Shamma, 2000).

Over the years, various robust and adaptive control techniques have been proposed to cope with such uncertainties. Nevertheless, only in recent years have advances in computational power and theoretical developments in adaptive control stability enabled the practical application of these methods to real-world aerospace systems.

$\mathcal{L}_1$ adaptive control represents a relatively recent advancement in adaptive control theory. It is specifically designed to provide fast adaptation while guaranteeing bounded system responses and robustness in the presence of uncertainties. A key characteristic of the $\mathcal{L}_1$ architecture is the decoupling of adaptation speed from robustness by explicitly limiting the control signal bandwidth through a low-pass filter, ensuring predictable transient behavior.

In this section, the implementation and results of $\mathcal{L}_1$ adaptive controller will be detailed. The work started from an open-source fixed-wing aircraft model (LozioAlce, 2025), which is adapted to represent the AIRSHIP A1 configuration. In addition, ground-effect-related modifications were introduced in the aerodynamic model, affecting lift and drag coefficients in order to better capture near-surface flight

characteristics.

After simulating the airship model, a baseline controller is designed using classical pole-placement techniques. The $\mathcal{L}_1$ adaptive controller is then introduced to compensate for parametric uncertainties and unmodeled dynamics within the actuator bandwidth. Parametric uncertainties are intentionally introduced to evaluate robustness, and controller performance is assessed through both deterministic simulations and Monte Carlo analyses with randomized parameter variations. These simulations are used to evaluate stability, transient behavior, and robustness of the flight control system under a wide range of operating conditions.

3.4.1 Baseline Controller

The baseline controller is designed to guarantee nominal system performance under ideal conditions, assuming no anomalies or parametric variations. It serves as a reference control law against which the robustness and performance of advanced controllers, such as the $\mathcal{L}_1$ adaptive controller, are evaluated.

The baseline control strategy adopted in this work is based on classical pole-placement techniques. Starting from the nonlinear aircraft model, a steady-state trim condition is first computed for a given operating point. The system is then linearized around this trim condition, and separate linear models for the longitudinal and lateral dynamics are obtained. These linearized models are subsequently used to design state-feedback controllers via pole placement.

Linearization around operating trim point

The trim condition is computed numerically by formulating the trim problem as a constrained optimization task. The objective is to find a steady-state combination of states and control inputs such that the time derivatives of the selected state variables are minimized, while satisfying physical and operational constraints.

Solve the constrained optimization problem using the MATLAB `fmincon` function to obtain the trimmed state vector x_{trim} and corresponding control inputs u_{trim}.

In our case, the trim condition is computed for a near-surface WIG operating point with a target airspeed of $V_{\text{trim}} = 28$ m/s and a center-of-gravity height above the water surface of $h_{\text{trim}} = 0.8$ m. The trim assumes straight and level flight, with zero roll and yaw angles ($\phi = 0,\ \psi = 0$), zero lateral velocity ($v = 0$), and zero angular rates ($p = q = r = 0$).

The optimization variables include the full state vector

$$x = [u, v, w, p, q, r, \phi, \theta, \psi, x_N, y_E, h]^T$$

and the control inputs

$$u = [\delta_e, \delta_a, \delta_r, \delta_t]^T.$$

Equality constraints are imposed to enforce the steady-state conditions listed above, while the remaining states and control inputs are allowed to vary freely during optimization. The optimization problem is solved using MATLAB's interior-point algorithm with tight convergence tolerances to ensure numerical accuracy of the trim solution.

3.4.2 $\mathcal{L}_1$ Adaptive Controller

The closed-loop system dynamics can be represented in the form

$$\dot{x}(t) = Ax(t) + B_m u(t) + f(x,t), \tag{61}$$

where $f(x,t)$ represents matched and unmatched uncertainties arising from modeling errors, disturbances, and cross-coupling effects. The total control input is defined as

$$u(t) = -K_{\text{baseline}} x(t) + u_{\text{ad}}(t), \tag{62}$$

where K_{baseline} is a nominal state-feedback gain designed using pole-placement, and u_{ad} is the adaptive control component.

A defining feature of the L_1 architecture is that the adaptive control signal is filtered before being applied to the plant:

$$u_{\text{ad}}(s) = -K_{\text{ad}} D(s)\, \hat{\eta}(s), \tag{63}$$

where $\hat{\eta}$ denotes the estimated uncertainty, K_{ad} is an adaptive gain, and $D(s)$ is a strictly proper low-pass filter that enforces a known control bandwidth. This filtering ensures that fast adaptation does not introduce high-frequency control action, thereby preserving robustness.

The L_1 adaptive controller is implemented as an augmentation to the baseline controller. Parametric uncertainties are intentionally introduced to evaluate robustness. Controller performance is assessed through both deterministic simulations and Monte Carlo analyses with randomized variations of aerodynamic and inertial parameters.

3.4.3 Simulation experiments

The simulation experiments focused on evaluating the ability of the $\mathcal{L}_1$ adaptive controller to maintain stable flight within and near the ground-effect regime, as well as to assess robustness under significant parametric uncertainty.

The aircraft model was initially trimmed inside ground effect at a center-of-gravity height of $h = 0.8$ m. All simulations start from this trimmed condition. The primary objective of the experiments is to maintain stable longitudinal flight in ground effect and to evaluate the controller response to altitude reference changes.

The simulation sequence is defined as follows:

- The vehicle starts at the trimmed altitude of 0.8 m.
- The altitude reference is increased by 0.2 m to evaluate small upward transitions within ground effect.
- The reference is then decreased by 0.5 m.
- Subsequently, a reference altitude of 10 m is sent to investigate whether stable flight outside the ground-effect region is achievable using the same trim and controller configuration.

- Finally, the altitude reference is returned to the original ground-effect operating point.

This sequence allows evaluation of controller behavior both inside and outside the ground-effect region, as well as during transitions between operating regimes.

3.4.4 Monte Carlo Robustness Analysis

To assess robustness, a comprehensive Monte Carlo simulation framework was developed. The analysis evaluates controller performance across a wide range of parameter variations representing manufacturing tolerances, aerodynamic model uncertainty, and operational variability.

All aerodynamic coefficients and physical parameters (with the exception of air density and gravity) were varied according to

$$P_{\text{MC}} = P_{\text{nom}} + \epsilon\, P_{\text{nom}} \cdot \text{rand}[0,1] \cdot \text{sign}(\mathcal{N}(0,1))\,, \tag{64}$$

where P_{nom} denotes the nominal parameter value and $\epsilon = 20\%$ represents the general uncertainty level.

Pitch Stability Derivative

The pitch moment derivative C_{m_α} has a critical role in longitudinal static stability. It was varied according to

$$C_{m_\alpha,\text{MC}} = C_{m_\alpha,\text{nom}} \cdot (0.5 + \text{rand}[0,1])\,, \tag{65}$$

ensuring that C_{m_α} remains negative while allowing its magnitude to vary by up to $\pm 50\%$ around the nominal value of -0.60. This constraint is necessary because positive values of C_{m_α} lead to pitch divergence, which cannot be stabilized by adaptive control alone.

Thrust Variations

Maximum available thrust was perturbed around the nominal value according to

$$T_{\text{max,MC}} = T_{\text{max,nom}} \cdot (1 + (2\,\text{rand}[0,1] - 1)\,\delta_T)\,, \tag{66}$$

with $\delta_T = 20\%$.

Input Cross-Coupling

Control surface cross-coupling effects were modeled as

$$\Omega_{\text{MC}} = I_{4\times 4} + \Omega_{\text{coupling}}, \tag{67}$$

where Ω_{coupling} is a 4×4 matrix with random elements scaled by 10%. This formulation represents mechanical linkage imperfections and aerodynamic interference between control surfaces.

3.4.5 Results

This subsection presents the results of the Monte Carlo simulations performed to evaluate the robustness of the $\mathcal{L}_1$ adaptive controller under parametric uncertainty. A total of 20 Monte Carlo runs were executed, each corresponding to a different randomized realization of aerodynamic parameters, inertial properties, thrust limits, and input cross-coupling, as described in the previous section.

In all result figures, the individual colored trajectories represent the outcomes of the Monte Carlo runs, while the black trajectory corresponds to the nominal case using the unperturbed parameter set. This reference trajectory is included to facilitate direct comparison between nominal and perturbed system behavior.

Fig. 21 shows the control inputs, including elevator, aileron, rudder, and throttle commands. These plots illustrate the variability of control effort required to maintain stability and tracking performance under parameter uncertainty. Despite the imposed variations, the control signals remain bounded across all Monte Carlo realizations, indicating that the adaptive controller effectively compensates for uncertainties within the actuator bandwidth.

Fig. 22 presents the key state variables relevant to longitudinal flight performance, including airspeed, angle of attack, pitch angle, and altitude. The results show that, although transient responses vary between Monte Carlo runs, all trajectories remain bounded and follow the commanded altitude profile without divergence or loss of stability.

Finally, Fig. 23 shows a zoomed-in view of the altitude response during the initial phase of the simulation, focusing on the portion of the experiment where the vehicle operates within the ground-effect region. This detailed view highlights the controller behavior during near-surface flight, where aerodynamic sensitivity is highest. The results show consistent altitude regulation across the Monte Carlo runs during this phase, with limited dispersion around the reference trajectory.

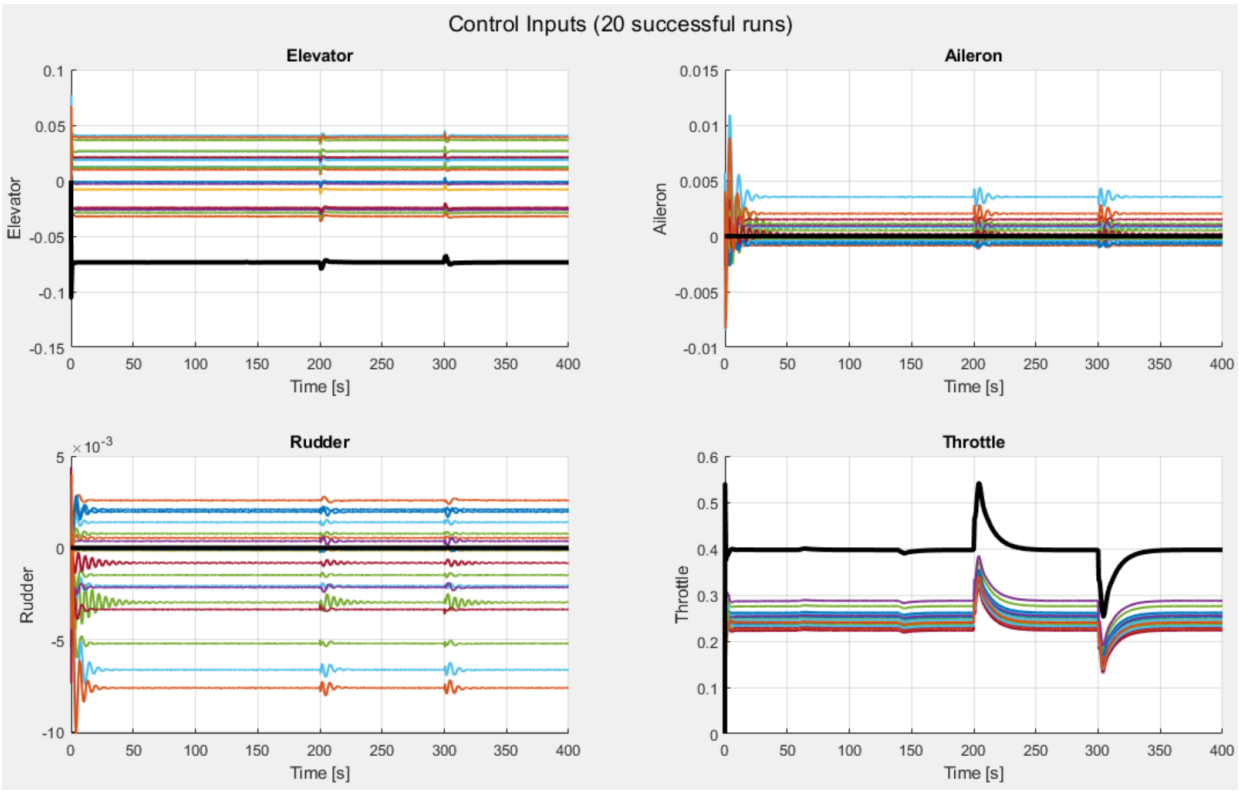


Figure 21: Monte Carlo simulation results (20 runs) showing control inputs. Colored lines correspond to Monte Carlo realizations with parameter uncertainty, while the black line represents the nominal case.

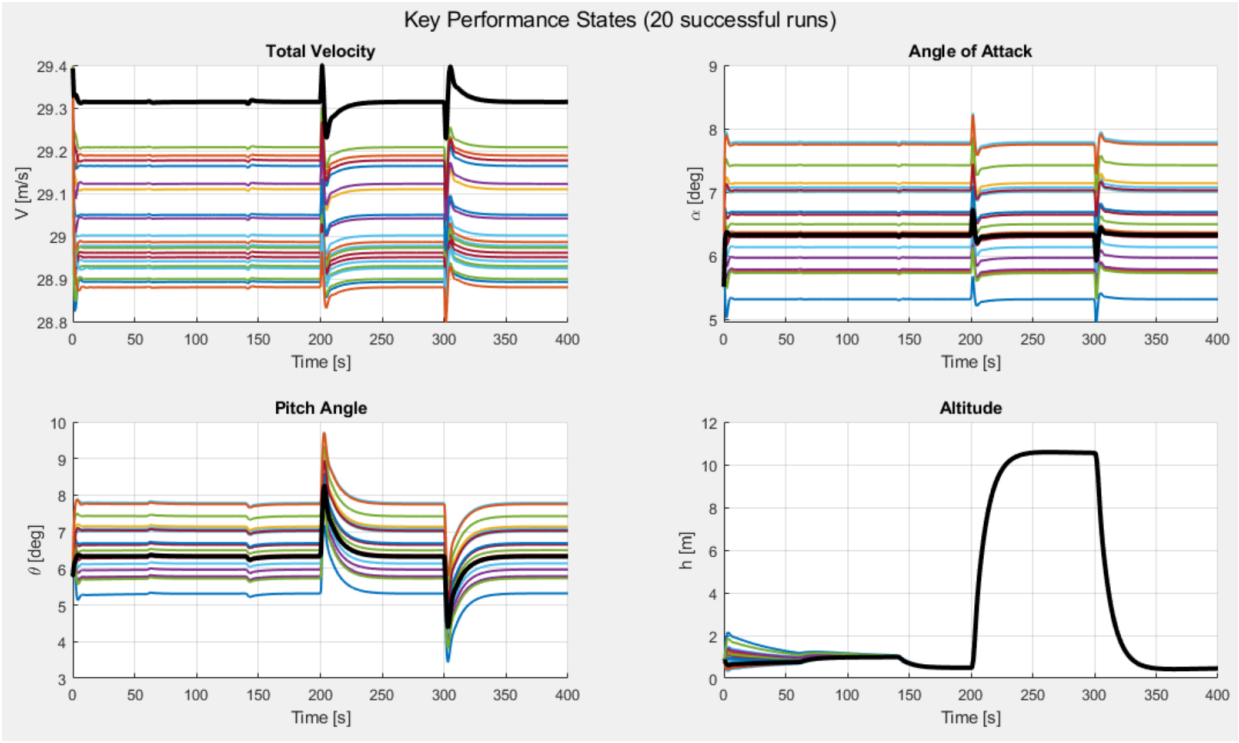


Figure 22: Monte Carlo simulation results (20 runs) showing key state variables: airspeed, angle of attack, pitch angle, and altitude. Colored trajectories correspond to perturbed parameter cases, while the black trajectory represents the nominal model.

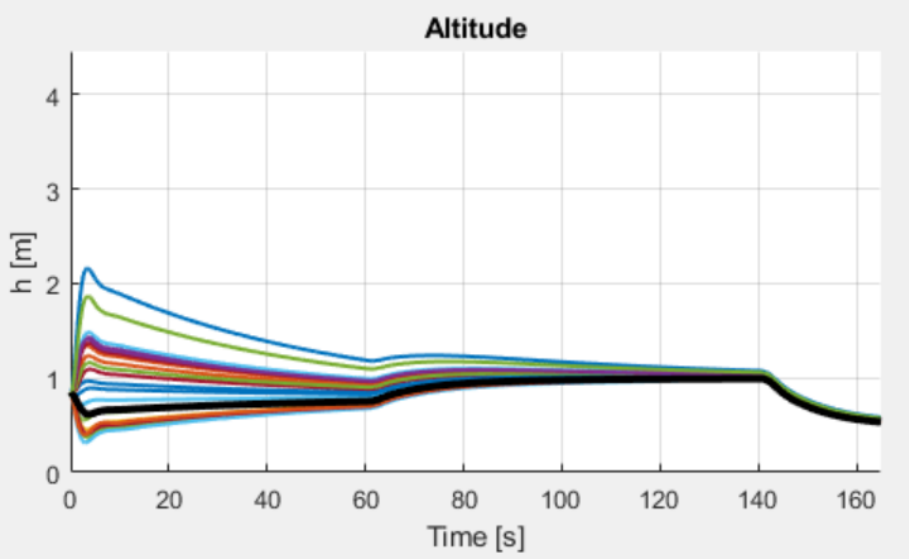


Figure 23: Zoomed view of altitude response during the initial ground-effect flight phase. The figure highlights controller behavior near the water surface, where aerodynamic sensitivity is highest.

Overall, the Monte Carlo results demonstrate that the $\mathcal{L}_1$ adaptive controller maintains stable flight and acceptable tracking performance in the presence of significant parametric uncertainty, both within ground effect and during transitions to higher-altitude flight. These properties make L_1 adaptive con-

trol a suitable candidate for WIG vehicle flight control, where operating conditions and aerodynamic characteristics can vary substantially and gain scheduling becomes impractical.

3.5 Fuzzy Control

Fuzzy logic is a mathematical approach to handling reasoning that involves approximate rather than exact reasoning. It was introduced by Lotfi Zadeh in the 1960s as a means to model reasoning that closely resembles human cognitive processes (Zadeh, 1988). Unlike classical binary logic, which deals with clear true or false (0 or 1) values, fuzzy logic deals with degrees of truth, allowing for greater flexibility and adaptation in control systems. In fuzzy logic, inputs and outputs are modeled using fuzzy sets and membership functions. These membership functions map crisp input values (such as the error in height or velocity) to degrees of membership in various fuzzy sets (such as ”small”, ”medium”, ”large”). The system then applies a set of rules to infer outputs (such as deflection of the control surface).

A typical fuzzy controller consists of the following main components:

- Fuzzification - The process of converting crisp input values into fuzzy values based on predefined membership functions
- Rule Base - A set of if-then rules, generally created by domain experts, that describe how to handle different fuzzy input values and map them to fuzzy output values.
- Inference Engine - The system that applies the rules to compute the fuzzy output based on the fuzzified inputs.
- Defuzzification - The process of converting the fuzzy output values back into crisp values, which can then be used by the system to perform an action

Fuzzy controllers have been widely applied in flying vehicles because they can handle the nonlinear, uncertain, and dynamic nature of aerodynamics without requiring an exact mathematical model. By mimicking human reasoning through linguistic rules, they can provide robust altitude, attitude, and trajectory regulation. Two common fuzzy inference approaches are Mamdani and Sugeno (Iancu, 2012; Nguyen et al., 2019). Mamdani controllers use fuzzy sets for both inputs and outputs, making them highly interpretable and suitable for visualizing how rules translate into control actions—an advantage when transparency and pilot-like reasoning are desired. Sugeno controllers, on the other hand, employ crisp linear mathematical functions for outputs, which makes them computationally efficient and easier to integrate with optimization or adaptive schemes, but less intuitive to interpret. In both cases, the design relies on membership functions that map numerical variables into linguistic terms. The choice of membership function type (triangular, trapezoidal, or Gaussian) affects the smoothness and sensitivity of the controller. For flying vehicles, Gaussian functions are often preferred because their smooth overlap ensures gradual transitions between control actions, reducing sudden changes in actuator commands and enhancing flight stability.

Having said that, this approach is well-suited for nonlinear systems with uncertain dynamics, such as WIG vehicles, which operate in the ground effect, so small changes in altitude can cause disproportionate shifts in lift and stability. By using fuzzy logic, the controller can accommodate these nonlinearities and uncertainties without requiring an exact mathematical model, ensuring altitude regulation and smooth surface deflections even under rapidly changing flight conditions.

3.5.1 Controller Design

In order to maintain the desired height with the fuzzy controller, the inputs were defined as the height error (difference between desired and actual altitude) and the rate of change of altitude (vertical velocity). For the controller output, the elevator surface deflection was chosen, which directly affects the vehicle's pitch and, therefore, its altitude.

The Mamdani fuzzy inference system was selected over the Sugeno for several reasons. Mamdani provides a more intuitive and linguistically transparent rule base, which is particularly useful when modeling pilot-like decision-making. Unlike Sugeno, which produces crisp outputs through weighted averages of linear functions, Mamdani maintains the interpretability of fuzzy sets throughout the inference process. This makes it easier to visualize how rules interact and to adjust them during tuning.

For the membership functions, each variable was represented by five linguistic terms: Negative Big (NB), Negative Small (NS), Zero (ZO), Positive Small (PS), and Positive Big (PB). These variables were chosen to provide sufficient resolution for capturing both large deviations and fine adjustments, while avoiding excessive complexity in the rule base. The symmetric structure around zero ensures balanced treatment of upward and downward deviations.

For membership functions, the Gaussian functions were used because of their smoothness and differentiability. Unlike triangular or trapezoidal sets, Gaussian functions avoid sudden changes at the boundaries, which reduces the risk of discontinuities in the control surface. This smooth overlap between adjacent terms ensures gradual transitions in the controller output, leading to more stable elevator deflections and improved altitude regulation. Additionally, Gaussian sets better reflect the probabilistic nature of sensor noise and uncertainty, making them well-suited for real-world flight conditions. The membership functions are shown in the Fig. 24.

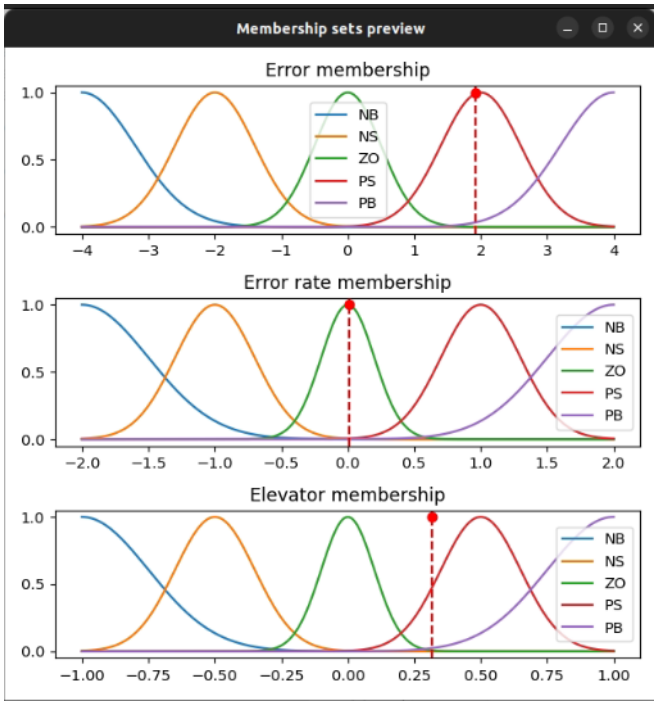


Figure 24: Membership Functions

The effectiveness of the fuzzy controller depends on a well-structured rule base that links our input

variables (height error and vertical velocity) to the output (elevator deflection). These rules are designed to capture intuitive flight behavior, ensuring that the controller responds in a way that resembles pilot decision-making under varying altitude conditions. The rule base chosen is intended to match pilot reasoning without a full aerodynamic model:

- When the height error is large, and the vertical velocity is negative (descending rapidly), the system responds by commanding a large upward deflection of the elevator (e.g., pull up) to arrest the descent.
- When the height error is large, and the vertical velocity is small or near zero, the system gradually adjusts the elevator to return the vehicle to the desired altitude by making smaller corrections (e.g., slight pull-up).
- When the height error is small but the vertical velocity is negative (descending), the system applies a gentle upward command (small pull-up) to correct the descent and avoid oscillation.
- When the height error is zero but the vertical velocity is positive (climbing), the system applies small downward deflections to prevent over-climbing.
- When both the height error and vertical velocity are near zero (on target and stable), the system does not apply any correction, allowing the vehicle to remain at the desired altitude.
- When the height error is positive (too low), and the vertical velocity is also positive (climbing too fast), the system applies a strong downward deflection to slow the climb and bring the vehicle back to the desired altitude.

Rules are evaluated using Mamdani min–max inference, and outputs are defuzzified via the centroid method. These rules ensure that the system adjusts the elevator deflection in response to altitude errors and vertical velocity. Also, the symmetric structure ensures consistent behavior around the setpoint.

An important feature of implementing fuzzy logic in this WIG vehicle control system is the dynamic adjustment of the error and error rate ranges. If the altitude error becomes too large, the fuzzy system can increase the tolerance (i.e., expand the fuzzy range) to prevent abrupt control actions. Vice versa, if the error is small, the system tightens its tolerance, ensuring better control and responsiveness. This is why adaptive scaling was implemented, which means that the range expands dynamically when vertical velocity or height exceeds a threshold. This ensures responsiveness during rapid climbs or descents. In addition, to further enhance the control signal and avoid sudden changes in elevator deflection, exponential smoothing has been applied to elevator command to prevent excessive oscillations or jumps.

3.5.2 Implementation

The controller was implemented using Python and X-Plane 12. A custom GUI was created for this purpose with the following capabilities:

- Modification of the Altitude Setpoint
- Modification of the Error and Error Rate Range
- Modification of the Simulation Duration
- Modification of the control rate limit
- Visualization of Membership Functions and the System’s Current State

- Live Plotting of Key Variables (height, velocity, elevator deflection...)
- Data Logging of Key Variables (height, velocity, elevator deflection...)
- Starting and Stopping the Simulation
- Choosing the Data Logging Path

The GUI is shown in Fig. 25.

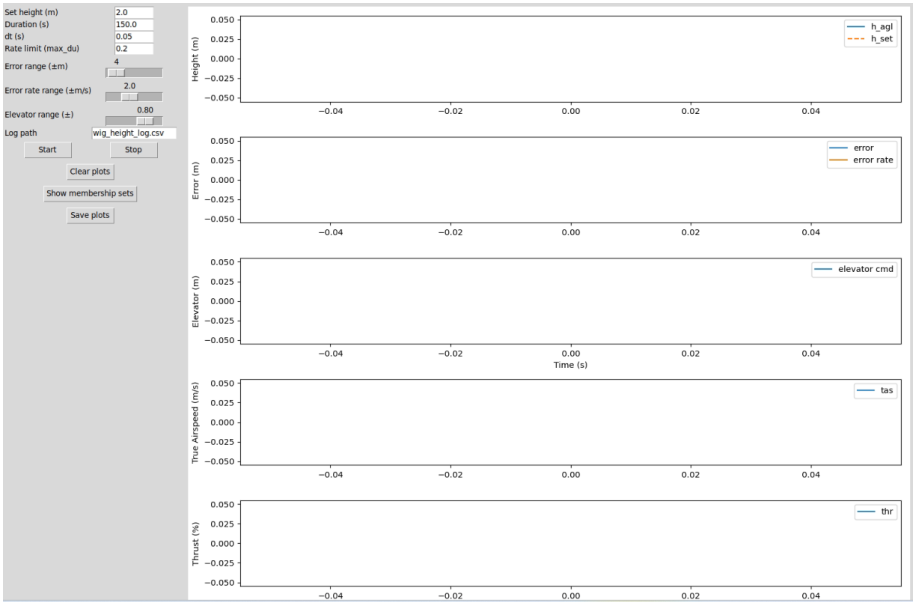


Figure 25: Custom Graphical User Interface

The X-Plane 12 simulator is used to model the flight dynamics of the AIRSHIP A1 prototype, handle control command transmission, and collect flight data, while Python is used for controller implementation, GUI design, and post-flight analysis.

3.5.3 Results, Evaluation and Conclusion

The simulation begins with the A1 prototype positioned on the water, with no control surface deflections or applied thrust, as shown in Fig. 26.

The parameters of the fuzzy control are set as follows:

- Error range is set to $[-4, 4]m$
- Error rate range is set to $[-2, 2]m/s$
- Elevator deflection range is set to $[-0.8, 0.8]$, where maximum values are $[-1, 1]$

After setting up everything, the simulation starts with the initial applied thrust of 50%. The simulation results are shown in the Fig.27a, Fig.27b, Fig.27c, and Fig.27d.

The True Airspeed profile (Fig. 27b) shows the vehicle accelerating from rest and reaching sufficient speed for lift generation, indicating a successful takeoff sequence. As expected, the airspeed increases



Figure 26: X-Plane 12 Simulation

steadily during the acceleration phase, corresponding to the applied thrust and gradual transition from displacement motion on water to aerodynamic lift-dominated flight.

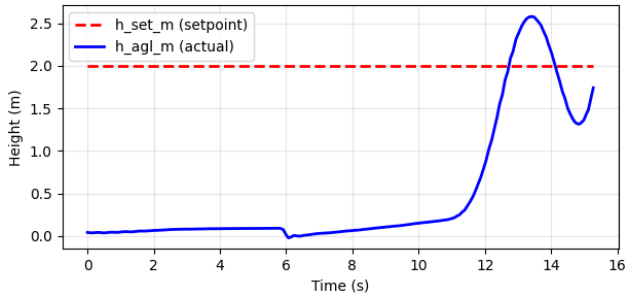
The altitude error evolution during takeoff (Fig. 27d) illustrates the dynamic nature of the lift-off process. As the vehicle leaves the water surface, the altitude error initially grows and then changes sign as the controller compensates for the increasing lift. The oscillatory behavior observed during this phase is characteristic of early ground-effect flight, where small variations in height lead to significant changes in aerodynamic forces. Despite these transients, the controller maintains bounded error and successfully stabilizes the vehicle above the water surface.

During takeoff, the WIG vehicle is commanded with 50% thrust and an elevator deflection of 0.3. Under these conditions, the aircraft gradually builds sufficient lift to leave the water surface in approximately 15 seconds. This takeoff profile reflects a moderate acceleration and lift-off behavior, without aggressive thrust or control surface inputs.

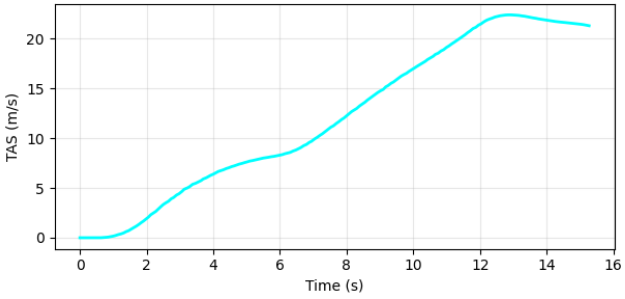
The use of limited thrust and moderate elevator deflection results in a smooth transition from waterborne motion to ground-effect flight, avoiding abrupt pitch changes or excessive control surface activity. It should be noted, however, that operation at low speed on water is associated with significant hydrodynamic drag, and therefore such a takeoff strategy does not necessarily minimize energy consumption. The suitability of this takeoff strategy, including its energy implications and sensitivity to environmental conditions, will be further evaluated during forthcoming field tests with the physical scaled prototype, where hydrodynamic effects, actuator limitations, energy consumption and sensor noise can be assessed under real operating conditions.

In conclusion, the use of fuzzy logic to control the WIG vehicle offers several advantages over traditional control methods. It provides a flexible and adaptive approach capable of handling the nonlinear and uncertain dynamics inherent in WIG vehicle flight. By using fuzzy sets and intuitive if–then rules, the system continuously adjusts its control outputs in response to altitude error, vertical velocity, and external disturbances.

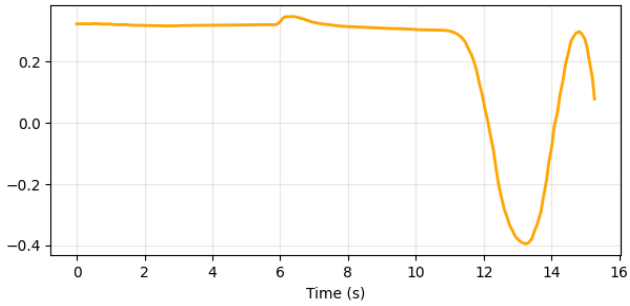
The ability to tune the controller parameters and maintain stable behavior in the presence of noise makes fuzzy logic a suitable candidate for AIRSHIP takeoff and low-altitude altitude regulation. Future



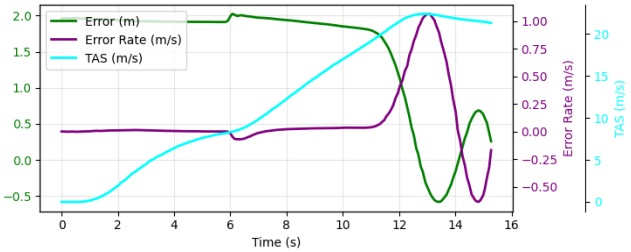
(a) Height setpoint vs. actual altitude



(b) True airspeed



(c) Elevator deflection



(d) Height error [m], error rate [m/s], and true airspeed (TAS [m/s])

Figure 27: Fuzzy control performance during takeoff.

work includes reducing oscillations through refinement of membership functions and input/output ranges, introducing additional stabilization mechanisms such as anti-windup strategies, and extending the fuzzy control approach to velocity, yaw, and roll regulation.

4 AI-based control

Reinforcement Learning (RL) offers a data-driven methodology for synthesizing feedback policies that optimize closed-loop performance through direct interaction with a dynamic environment. Unlike classical linear and non-linear control techniques that rely on explicit analytic models and assumptions about system structure, RL leverages sequential decision-making under uncertainty to learn policies that maximize long-term reward signals. For autonomous aerial systems, this allows controllers to reason over non-linear, coupled dynamics, actuator limits, and delayed effects, while potentially reducing the need for manual gain tuning or high-fidelity analytic modelling.

In recent years, model-free and model-based RL algorithms have demonstrated promising results for robotic manipulation, locomotion, and autonomous vehicle control. Their application to aircraft systems remains an emerging field, where challenges include safety, sample efficiency, stability guarantees, and deployment on real hardware. Nevertheless, RL offers a complementary approach to the classical frameworks presented in section 3. This approach has the potential to outperform hand-tuned control laws when dealing with aerodynamic non-linearities, ground effect, and uncertain flight regimes.

This section focuses on DreamerV3, a state-of-the-art model-based algorithm in which world models are learned to enable efficient long-horizon reasoning in latent state spaces.

4.1 Learning-Based Control with DreamerV3

DreamerV3 is a general and scalable model-based reinforcement learning algorithm that learns a compact predictive world model and uses it to optimize agent behaviour via imagined rollouts in the latent space. The approach is built on the principle of learning a differentiable internal model of environment dynamics that can be used to simulate future outcomes and thereby support long-term planning without requiring extensive interactions with the real environment. By leveraging robustness techniques such as normalization, balancing, and input transformations, DreamerV3 exhibits stable learning across a wide variety of domains and tasks without the need for domain-specific hyper-parameter tuning.

One of the most notable empirical results demonstrated in the original work (Hafner, Pasukonis, Ba, & Lillicrap, 2025) is that DreamerV3, with a single configuration of hyper-parameters, significantly outperforms specialized reinforcement learning methods across more than 150 diverse benchmark tasks. The algorithm's generality is further illustrated by its ability to solve challenging exploration problems. These characteristics make DreamerV3 an attractive candidate for complex continuous control problems where dynamic models are available but manual tuning of controllers may be difficult.

A DreamerV3 agent was trained to perform altitude regulation. The task consists of maintaining flight along a forward trajectory while matching a time-varying altitude reference. Fig. 28 shows the learned behaviour over multiple rollouts, with 3D trajectory tracking, orientation stabilization, position evolution, and linear velocity regulation. The results indicate stable control behaviour 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 11, demonstrating tracking accuracy in the sub-meter range with energy usage that can be improved.

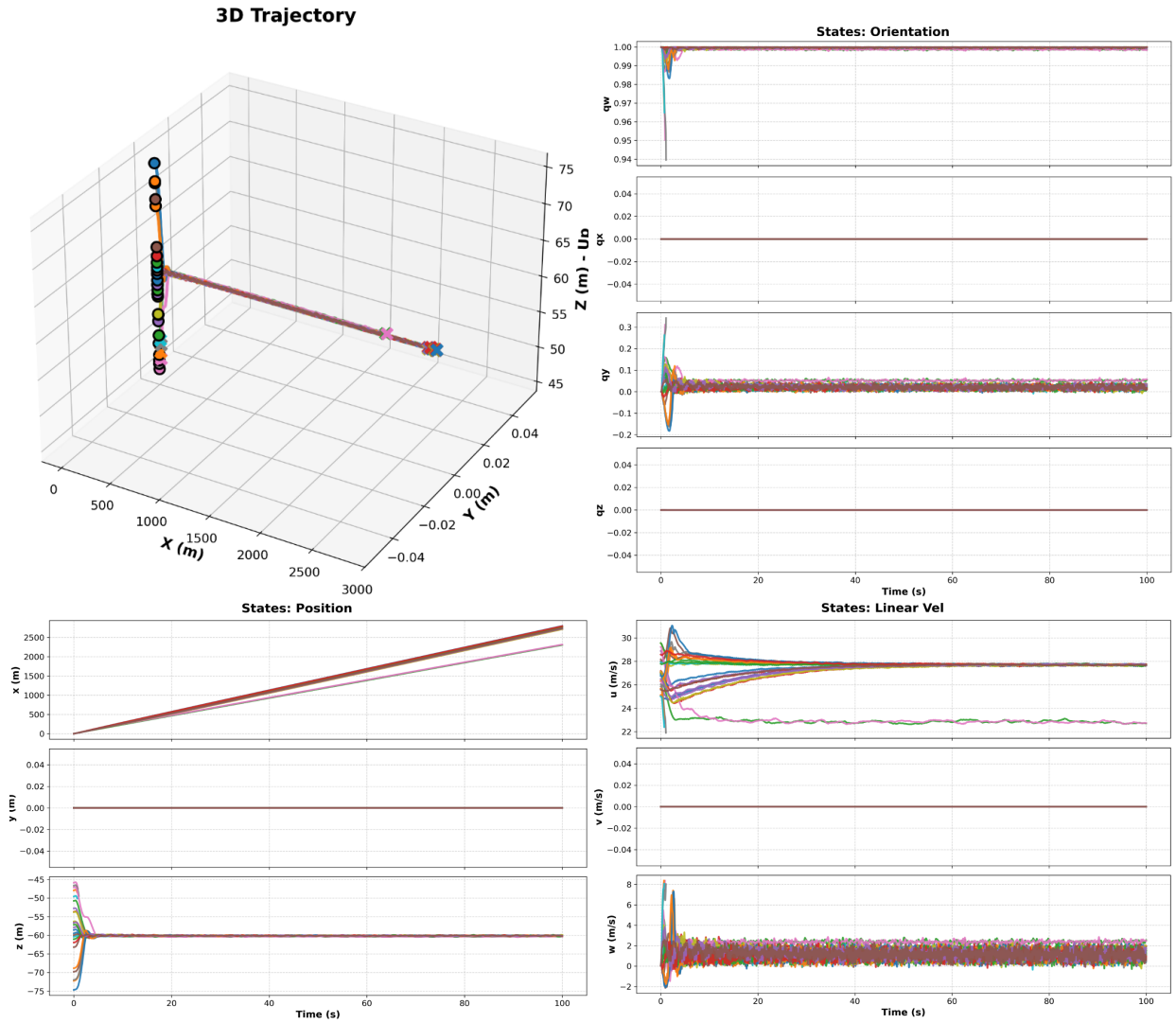


Figure 28: DreamerV3 agent controlling the airship along a dynamic altitude-keeping trajectory. Top-left: 3D trajectory tracking. Top-right: orientation convergence. Bottom-left: position over time. Bottom-right: body-frame velocity components.

5 Simulation frameworks

This section describes the simulation frameworks employed to assess the proposed control strategies and validate aircraft performance. Three independent simulation environments were used to ensure consistency and cross-verification. Two custom simulators, one implemented in *MATLAB/Simulink* (Inc., 2022) and one in *Python*, which share an identical set of aircraft dynamic equations and aerodynamic models (described in section 2), enabling code-level equivalence and repeatability across platforms. In addition, a high-fidelity environment was constructed by coupling a commercial aircraft simulator *X-Plane 12* (Laminar Research, 2024) with the *Gazebo* robotics simulator to incorporate realistic flight dynamics, sensor effects, and hardware-in-the-loop capabilities. The use of multiple simulation frameworks permits both efficient algorithm development and validation of control systems across.

Table 11: Mean ± standard deviation across five DreamerV3 runs for dynamic altitude tracking.

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

5.1 Python framework

The Python 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 consists of two primary modules: the *agent* and the *environment*.

The agent module supports a wide range of control models, including classical approaches such as Linear Quadratic Regulator (LQR), Linear Quadratic Regulator with Integral action (LQRI) and planned future addition of PID, 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.

The environment exposes a compliant Gymnasium (Towers et al., 2024) interface through a Python `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.

The developed framework also provides integration with Stable Baselines3 (SB3) (Raffin et al., 2021), 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.

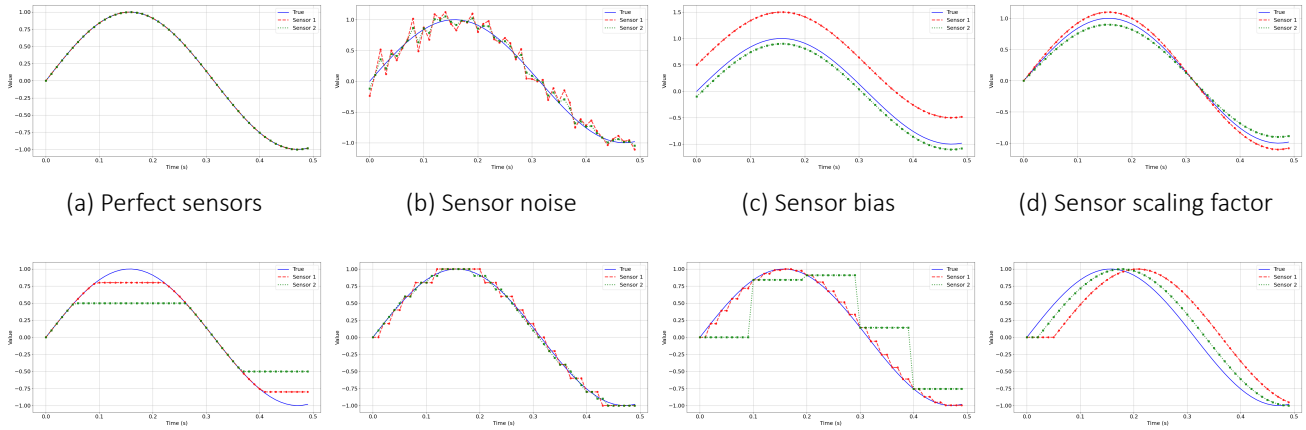
The environment module supports multiple simulation back-ends and interoperation with external tools such as *OpenVSP* (NASA OpenVSP Team, 2025), *MATLAB* and *X-Plane 12*, allowing flexibility and enabling various controller design workflows.

The environment simulates full 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 (see section 2.1.2).
- *Actuators*: Control surface deflections and thrust values are passed through first- or second-order actuator dynamics (see section 2.2), allowing simulation of latency, saturation, and rate-limited responses.
- *Environmental Effects*: Wind gusts, turbulence fields, and atmospheric density gradients are in-

jected into the dynamics via different noise models, enabling robustness testing under realistic conditions.

- *Sensors*: A developed onboard sensor model class can simulate IMU measurements (accelerometer, gyroscope), GNSS, and optional encoders. Sensor noise, bias, scaling, saturation, sampling rate, resolution, or delay can be individually or in combination added to evaluate controller performance under degraded sensing (see Fig. 29).
- *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.



(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 29: Illustration of different sensor effects implemented in the Python simulator.

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.

The framework supports rapid prototyping of different AIRSHIP airframes evolutions and modifications via JSON-based configuration files. Each aircraft model 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 mentioned in section 2.1.1.

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.

To improve validation and high-fidelity visualization, an interface to the X-Plane using the open-source Python *XPlaneConnect* API (NASA Ames Research Center, 2025) developed by NASA was added. If given the same aircraft physical properties (geometry and flight initial conditions), trajectories generated in the simulator can be replayed or compared within X-Plane's physics 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 environ-

ment, 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 water locations.

5.2 MATLAB/Simulink framework

In addition to the Python framework, MATLAB and Simulink provides a graphical, block-diagram–based modelling paradigm that enables rapid prototyping of control architectures and plant models. It supports efficient design iteration, parameter tuning, and automated testing of any system dynamics.

This simulator is used to design and validate the baseline control laws before integrating them into the Python-based framework developed. Since Simulink includes extensive libraries for signal processing, flight dynamics, control systems and ODE solvers, it accelerates the configuration and evaluation of candidate control strategies with minimal implementation overhead. The rapid development workflow allows the testing various control concepts without writing lower-level code, reducing development time.

The MATLAB simulator also serves as a way to validate if the physics models across both frameworks converge under equivalent conditions, increasing confidence that both implementations correctly represent the intended dynamics, reducing risk of modelling errors before hardware deployment.

Lastly, it also allows symbolic derivation of system matrices for LQR, LQRI and linear model identification.

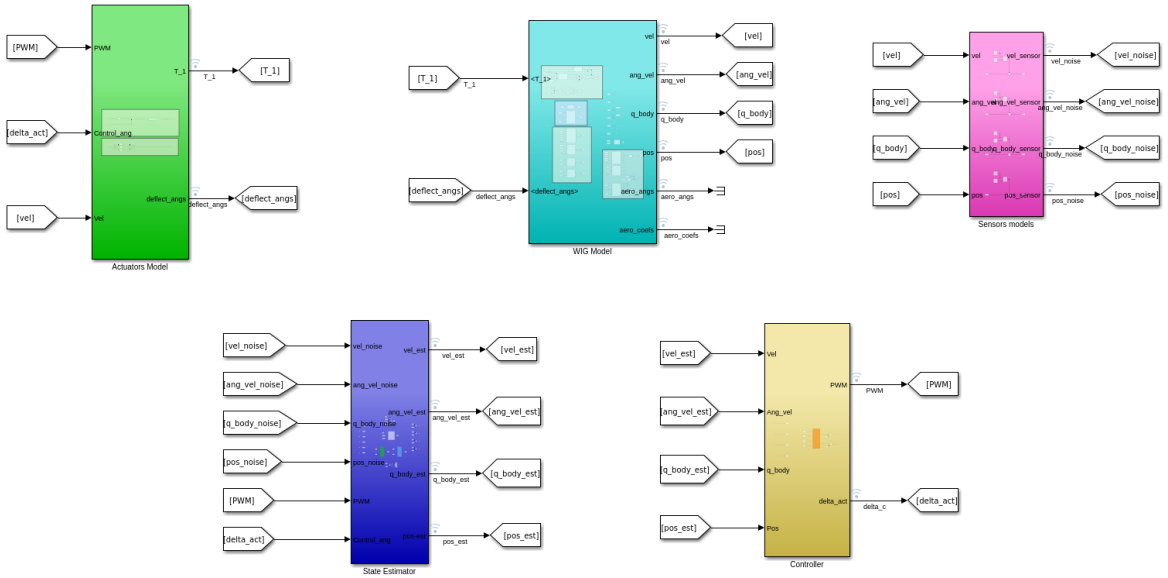


Figure 30: Block-diagram representation of the implemented aircraft dynamics and control architecture in Simulink, illustrating the modular structure of the simulation framework, with separate subsystems for actuator dynamics, sensor models, vehicle dynamics and feedback control laws.

5.3 X-Plane 12 framework

A reliable system model is essential for the development and testing of control algorithms. Several flight simulators are available for evaluating autonomous flight control systems. JSBSim is an open-source flight dynamics simulator featuring a physics-based modeling approach (Berndt & the JSBSim



Figure 31: AIRSHIP prototype designed in PlaneMaker, a proprietary software that comes with X-Plane 12.

Development Team, 2011). FlightGear is an open-source flight simulator that relies on JSBSim as its underlying flight dynamics engine (Perry, 2004). Microsoft Flight Simulator provides visually realistic graphics and detailed terrain representation, but is primarily focused on pilot training and visualization rather than control-oriented research (*Microsoft Flight Simulator*, 2025).

X-Plane, developed by Laminar Research, is equipped with a physics engine well-suited for testing control algorithms. Unlike simulators based on stability-derivative models, X-Plane utilizes Blade Element Theory to compute aerodynamic forces, enabling detailed modeling of aerodynamic effects across a wide flight envelope. X-Plane provides access to a large set of physical and state variables, a relatively simple user interface, and real-time data exchange capabilities, which make it suitable for research-oriented applications. The latest version, X-Plane 12, is FAA-certified for specific training applications (Yalçin & Ersoy, 2018).

In the AIRSHIP project, X-Plane 12 has been used as the primary simulation software for validating control algorithms. The NASA plugin XPlaneConnect provides the communication interface between X-Plane 12 and ROS 2 architecture, using the UDP protocol for data exchange. (NASA, n.d.).

By default, X-Plane includes a variety of off-the-shelf aircraft models, allowing users to get started with the real-time flying simulation experience. These models cover a wide range of aircrafts and several seaplanes. While seaplanes are the closest available options for testing near-surface flight operations, they do not fully capture the dynamics of WIG vehicles. For this reason, it was necessary to develop a custom WIG craft model, see Fig. 31. The model was created using Plane Maker, the aircraft design tool that comes along with the simulator. The design and modelling process combined Computational Fluid Dynamics (CFD) analysis, mathematical modelling, and detailed measurements of our project prototype. The 3D render data and blueprints were used to ensure that the simulated vehicle closely resembles our actual prototype. This design work carried out within the AIRSHIP project is being done by Tampere University. The validation and initial testing of the prototype done in the X-Plane 12 can be seen in Fig. 32.

5.3.1 System Architecture

ROS 2 Humble middleware has been used as the primary message-passing interface and communication underlay, ensuring modularity and scalability of the overall architecture. To achieve seamless integration between subsystems, several custom message definitions were developed. These custom messages played a crucial role in improving system transparency, facilitating intuitive understanding of inter-node

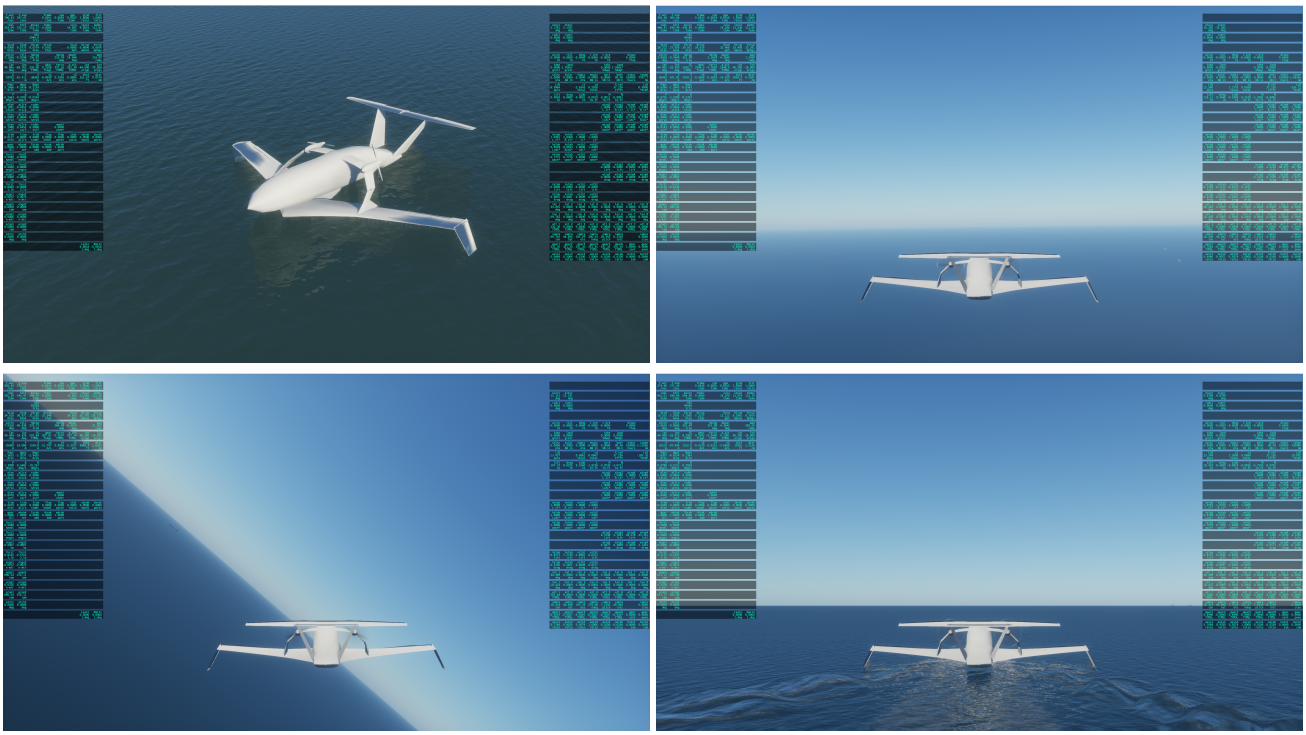


Figure 32: Prototype tested in X-Plane 12.

communication, and enabling rapid debugging throughout the development and testing phases.

Given that the proposed model is a low-altitude flying WIG craft, the controller required gain adjustments during the development and testing phase to ensure stable operation. For this purpose, a live tuning mechanism was implemented, allowing the PID gains to be modified interactively while the WIG craft is operating in simulation. This feature is intended exclusively for controller development, testing, and tuning, and enables efficient parameter adjustment under varying simulated flight conditions.

The overall communication architecture illustrating the flow of messages within the ROS 2 ecosystem, along with its interaction with X-Plane 12 and the control loop, is depicted in Fig. 33. The architecture shows the integration using a cascaded PID controller as a representative example. The same software and communication framework were also used to integrate and evaluate alternative control strategies, including LQG and fuzzy logic controllers, without requiring structural changes to the architecture.

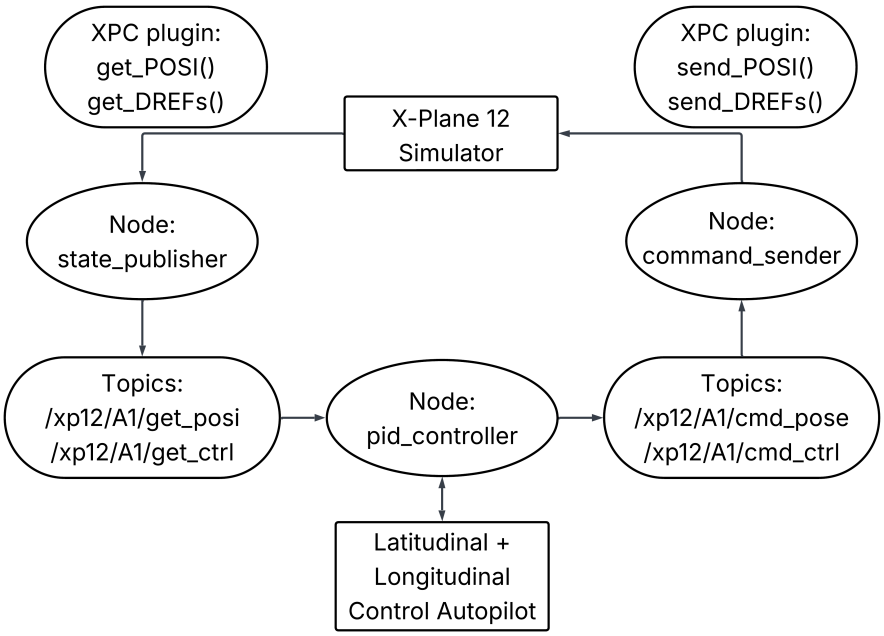


Figure 33: ROS 2 - X-Plane 12 closed-loop control architecture.

6 Control Software Architecture

To enable the integration, execution, and validation of the control algorithms developed under Task 5.2, a dedicated control software architecture is required. This section presents the control software architecture developed to support autonomous operation of high-speed WIG vehicles. This work has been done in collaboration between UPM and Tampere University and has been published at the IEEE 13th International Conference on Control, Mechatronics and Automation (Ilić et al., 2025).

Unmanned WIG vehicles rely predominantly on autonomous control, navigation, and communication systems to ensure safe and stable operation in low-altitude, high-speed environments. Consequently, the control software architecture must be scalable, fault-tolerant, and capable of real-time processing, while supporting integration with ground control infrastructure. These requirements exceed those typically addressed by adaptations of conventional UAV software frameworks and motivate the development of a dedicated, WIG-oriented architecture.

The work presented in this section reviews the system requirements, introduces the custom Arm-In-Ground (AIG) hardware platform developed by Tampere University, and details the adopted open-source software stack. The architecture integrates ROS 2 as middleware, Pixhawk-based flight controllers, ArduPilot autopilot firmware, MAVLink communication, and Mission Planner as the Ground Control Station (GCS). The proposed architecture has been evaluated through HIL testing to assess system performance, timing behaviour, and computational resource usage under different command update rates.

The AIRSHIP commercial concept targets a full-scale WIG vehicle with a wingspan of approximately 25 m and a maximum take-off weight (MTOW) of about 16 t. The control software architecture described here is designed to support this class of vehicle, while being validated on a scaled 5 m wingspan technology demonstrator with an MTOW of approximately 110 kg. The use of modular, open-source components enables straightforward scaling across vehicle sizes without fundamental architectural changes.

The high-speed operational profile of WIG vehicles imposes strict latency and reliability requirements on the software. Sensor data must be processed at frequencies ranging from 100 Hz to 1 kHz, while actuator command loops must operate at a minimum of 10 Hz to maintain stable flight dynamics. The proposed architecture is designed to meet these timing constraints while maintaining robustness to communication delays and partial subsystem failures.

Finally, WIG vehicles occupy a unique regulatory position. According to IMO guidelines (International Maritime Organization, 2018), WIG vehicles operating in ground effect are classified as maritime vessels rather than aircraft. This classification influences software architecture decisions, as the system must comply with maritime operational reliability standards without the full burden of aviation software certification requirements. This regulatory framework allows the controlled use of open-source components during development, while maintaining appropriate safety and reliability margins for future commercial deployment.

6.1 Development

In this section, the development steps of the software architecture design at the system level and in more detail at the in-vehicle level are outlined, which could enable autonomous high-speed WIG vehicles. First, the system requirements that guided the design process are outlined, followed by the hardware architecture used, and finally, the software architecture, which combines open-source firmware

and modular design principles to enable scalable and autonomous operation is presented.

6.1.1 System Requirements

When designing the software architecture, it is important to define both functional and non-functional system requirements (Nouman & Azam, 2023). Having in mind the high-speed nature of WIG vehicles, the software should provide stability management and precise flight control(Dreus, Alekseyenko, Kulyk, & Nekrasov, 2025). Accurate real-time control of pitch, roll, and altitude, to maintain low-altitude flight within the ground effect, is crucial. Additionally, it is important to ensure safe flight regimes and transition between them. Obstacle avoidance is also important, so strong navigation and guidance systems are essential. This requires integrating data from sensors like GPS, IMU, radar, LiDAR, and altimeters. Real-time communication and telemetry are critical for reliable data exchange between the WIG vehicle and the ground station. Non-functional requirements include timing constraints for sensor processing, actuator commands, and the real-time performance of the whole system, such as fault tolerance, safety mechanisms, redundancy, and the ability to handle system failures. Modularity and scalability are necessary to support the development of mission capabilities, WIG vehicle dimensions, and payloads. The software should be robust to challenging environmental conditions, such as wind gusts, high waves, and sea spray. Integration requirements include sensor fusion techniques for accurate state estimation, reliable low-level control interfaces to manage servos and motors, and efficient resource management to handle operations. The software architecture should support different levels of autonomy, from human-in-the-loop piloting to full autonomy with adaptive route planning, diagnostics, and health monitoring systems. In the WIG vehicles scenario, the integration with ground control interfaces is vital for mission planning, real-time monitoring, and human operator intervention.

To summarize, several critical system requirements should be kept in mind when designing a software architecture that enables autonomy of the high-speed WIG vehicle. The system must provide real-time communication and telemetry with high responsiveness, while maintaining robustness to environmental conditions such as waves, wind gusts, and wind. Fast data processing capabilities are essential, along with support for different levels of autonomy. The architecture should incorporate modularity and scalability for future adaptability, while ensuring robust handling of system failures and secure data transmission.

6.1.2 Hardware Architecture

Before diving into the software architecture design and to get a clearer picture of the entire system, it is worth mentioning the hardware setup that was chosen and used in this research - a custom Printed Circuit Board (PCB) ARM-In-Ground (AIG) developed by Tampere University(Hoisko, 2025). The WIG's avionics system is built around distributed smart nodes that handle local computation, sensor interfacing, and actuator control. Each node is based on the ARM RK3328 System-on-Module (SoM), an open hardware platform that runs Linux, providing sufficient processing power for real-time decision-making and offering modular expandability for larger prototypes. The SoM sits on the carrier board AIG that enables high-speed fiber optic communication via SFP modules, interfaces for RS-485, I^2C, TTL UART, and copper Ethernet, as well as power management from either 28V aircraft supply or 6S LiPo batteries. The board integrates hardware-level protections for over-current and short circuits, and its four-layer design ensures signal integrity, EMI mitigation, and efficient thermal management, providing a reliable foundation for distributed autonomous control. The AIG is shown in Fig. 34.

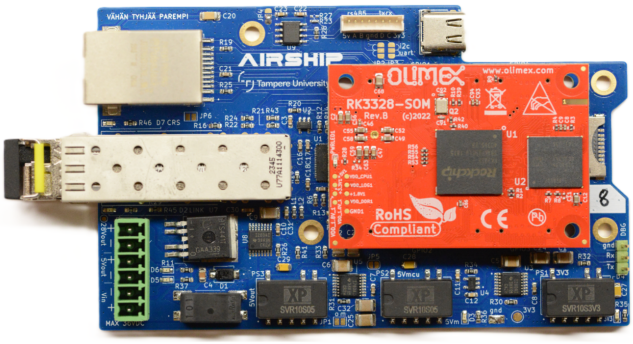


Figure 34: Top view of the AIG printed circuit board (blue) populated with the RK3328 System-on-Module (red), used for real-time embedded control.

6.1.3 Software Architecture Design

This subsection presents the proposed software architecture for autonomous WIG systems, emphasizing modularity, scalability, and open-source compatibility for seamless hardware integration. The design targets a 25-meter concept vehicle and will be validated on a 5-meter prototype.

Starting with the middleware layer of our architecture that is compatible with the AIG platform, ROS2 (Macenski, Foote, Gerkey, Lalancette, & Woodall, 2022) was chosen, an open-source middleware framework designed for real-time, distributed robotics applications. This framework has a modular node-based architecture that supports scalable and maintainable system design, which is essential for managing the integration of perception, navigation, and control in the autonomous WIG vehicles. The ROS2 Humble version was chosen because it offers improvements compared to the older versions, such as enhanced DDS-based communication, improved security features, better support for embedded platforms, and a longer end of life. Compared to the newest version, Jazzy, Humble was still chosen as a long-term support (LTS) release, which has a broader driver and package support, is compatible with other LTS releases, and has more predictable behaviour across systems.

Additionally, this version is compatible with microcontrollers and real-time operating systems, which aligns with the constraints and requirements of high-speed WIG vehicles. Furthermore, ROS2 Humble's support for real-time and safety-critical operations makes it a good choice to fulfil the previously stated requirements of WIG vehicles (Chovet et al., 2025).

To ensure robust and scalable communication across distributed nodes, a centralized ROS2 discovery server is employed, along with a backup instance. This setup improves network reliability and startup consistency. The backup server provides fault tolerance, allowing seamless recovery in case of primary server failure. More importantly, the discovery server architecture significantly reduces the communication complexity from $O(n^2)$ to $O(n)$, where n is the number of nodes. In default peer-to-peer discovery, each node must independently discover and maintain connections with every other node, leading to quadratic scaling and increased overhead. By centralizing discovery, each node only needs to register with the server, which then manages and propagates connection metadata. Complementing ROS2, for the flight controller the Pixhawk 6X (L. Meier, P. Tanskanen, L. Heng, G. Lee, F. Fraundorfer, and M. Polle feys, 2012) was chosen, which has high computational power, redundant sensor inputs, and many connectivity options, specifically the built-in Ethernet port.

For the 5-meter technology demonstrator, the open-source ArduPilot(ArduPilot Development Team,

2009) firmware ecosystem is employed because it provides flight control algorithms, mission planning functionalities, and built-in fail-safe mechanisms proven across a range of aerial platforms. This selection enables rapid prototyping and algorithm validation during the development phase. However, for the 25-meter wingspan vehicle, the architecture is designed to accommodate a commercial-grade, certified flight controller to meet safety standards, regulatory compliance, and mission-critical reliability requirements. The modular nature of the MAVLink communication protocol ensures seamless integration regardless of the flight controller implementation. The initial flight controller used is the fixed-wing aircraft controller from Ardupilot. Specifically, the stabilize mode is used, and the first validation provided satisfactory results. In parallel, custom-made controllers are being developed to be implemented instead of the available Pixhawk controllers.

To match the ArduPilot firmware, for the GCS, Mission Planner (ArduPilot Development Team, 2019) is used as compatible software. Mission Planner enables intuitive mission design, real-time telemetry monitoring, and diagnostics that are critical for validation and safe operation during both development and deployment phases. For real-time communication between the flight controller and the companion computer, previously mentioned AIG, Ethernet with MAVLink (Lorenz Meier, 2009) protocol was chosen as an efficient messaging standard that ensures effective command and control. Moreover, MAVROS, a ROS2 interface for MAVLink-enabled autopilots was implemented, which further bridges high-level ROS2-based modules and low-level flight controller functions. To ensure low-level code reliability, software for the application programming interface towards the higher-level control is separated from the low-level hardware control. As presented in Fig. 35, the motors, servos, navigation lights, and other physical components are controlled by independent services. AIG-API communicates with those services via UNIX sockets.

UNIX sockets act as an inter-process communication mechanism that allows processes to communicate with each other on the same machine. They provide a fast, efficient way for applications to exchange data locally without the overhead of network communication. UNIX sockets' efficiency and simplicity make them a good choice for WIG vehicles because they require fast and reliable communication between processes on the same host. An integrated flight recorder for reliable data logging and post-flight analysis has high-speed data acquisition, making it suitable for WIG vehicle operations. It is compatible with a wide range of sensors and open-source firmware, so it provides a flexible configuration and easy integration with the rest of the components of our software architecture. The sensor choice, fusion, and connection are developed by INESC TEC under WP5. Additionally, the onboard storage and real-time clock enable precise timestamping and secure data archive, which are crucial for diagnostics, performance evaluation, and safety assurance. Together, these technologies, when incorporated, provide a well-supported, extensible, and scalable software architecture for the WIG vehicle.

The high-level software architecture diagram is presented in Fig. 35. In the high-level software architecture diagram, the main components of our design proposal and the communication between them are illustrated. At the top level, the GCS, which in this case is implemented through Mission Planner, establishes a wireless communication link with the Flight Controller. This connection enables mission planning, parameter configuration, and real-time monitoring of flight data. The flight controller itself serves as the central processing unit onboard the WIG vehicle, responsible for executing autonomous missions and maintaining communication with the pilot's RC controller for manual override and command input. In addition, the flight controller continuously gathers and processes sensor data, ensuring stability, navigation, and safety throughout the mission.

A detailed breakdown of the low-level software architecture, which is set up within the AIG-API, is presented in Fig. 36.

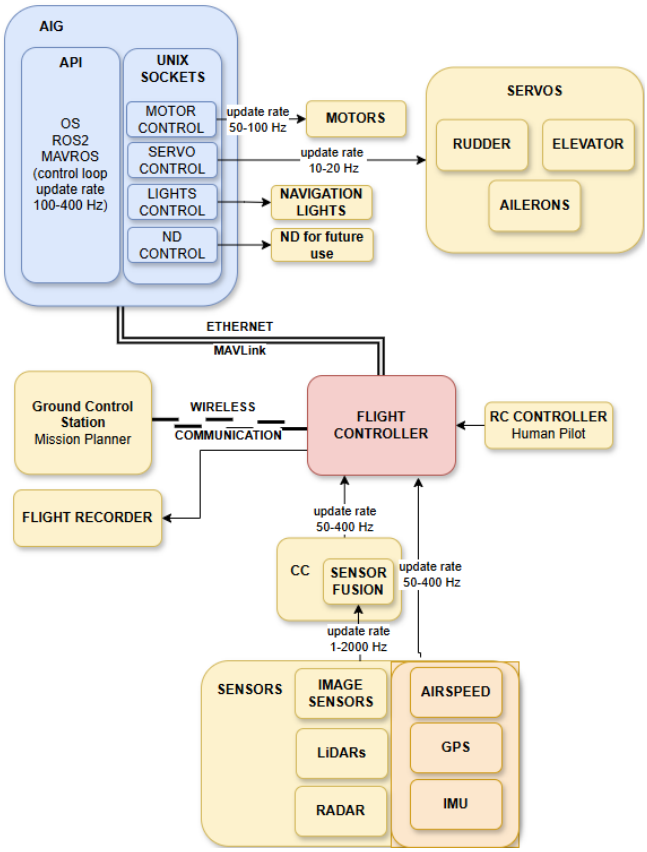


Figure 35: High-Level SW architecture diagram. The flight controller manages sensors, telemetry, and control commands, while the AIG-API layer provides ROS2/MAVROS middleware and Unix socket services for motors, servos, and auxiliary systems.

Within the low-level software architecture, three custom ROS2 nodes were developed to address the primary operational needs of the system: sensor data logging, control commands, and condition monitoring. The sensor data logging node subscribes to MAVROS node topics that broadcast flight information such as vehicle state, position, orientation, velocity, and battery status. This data is sent to custom ROS2 topics designed for power management and condition monitoring. This design ensures that sensor data is both saved for post-mission analysis and distributed in real time to components that maintain WIG's operational. The control command node acts as the bridge between the pilot's RC inputs and an onboard autonomous controller and the actuators responsible for maneuvers. It subscribes to the MAVROS flight command topics and translates them into low-level messages, which are then sent through the dedicated Unix Sockets. By decoupling these commands from the ROS2 middleware and connecting them directly to hardware interfaces, the system isolates faults, meaning that even if one software subsystem experiences delays or interruptions, the core control functions remain operational.

Previously mentioned Unix Sockets provide direct control over physical components - motors, servos, and navigation lights. These sockets serve as communication interfaces between the ROS2 nodes and are essential for ensuring precise actuation and reliable system control. These Unix Sockets are hosted within the AIG's operating system, yet placed outside of the ROS2 framework. This separation creates a hybrid design balancing modularity and robustness: ROS2 provides a flexible and extensible environment for autonomy and high-level control, while Unix Sockets provide independence and reliability.

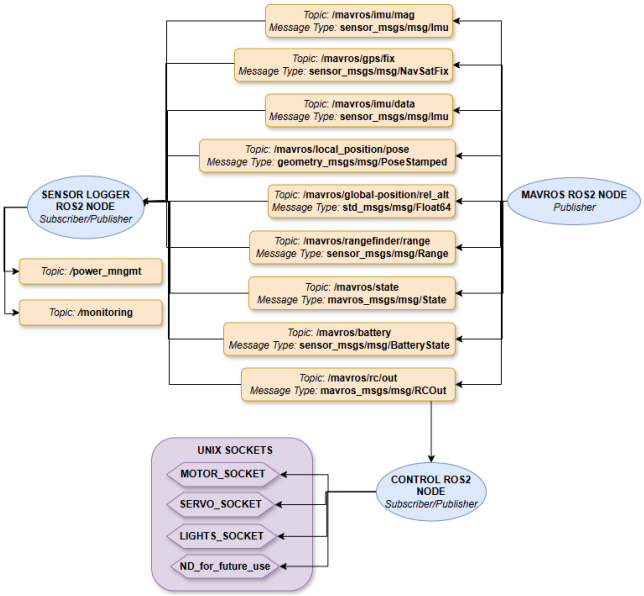


Figure 36: Low-level SW architecture within AIG-API. ROS2 nodes handle sensor logging, monitoring, and control, Unix sockets ensure direct, reliable actuation of motors, servos, and navigation lights. The diagram reflects the actual ROS2 topic structure.

Such an approach reflects the principle of designing a system in which no single component is solely responsible for the mission's success.

In addition to robustness, this software architecture is adaptable. The design idea was for the architecture to be plugin-and-play, allowing it to be reused, modified, and extended across different WIG prototypes without requiring major changes. Currently, this software architecture is being tested and validated on a 5-meter prototype platform. Insights from these tests will be used as information for refinements of the architecture, which is intended to be extended for prototype experimentation and scaling into the commercial WIG vehicle with a 25-meter wingspan.

6.2 Experimental Evaluation and Results

HIL tests validated software performance on the AIG board running ROS2 Humble with Fast DDS, connected to Pixhawk 6X via Ethernet using MAVLink/MAVROS. Command rates from 10–1000 Hz were evaluated for synchronization, CPU load, memory usage, and bandwidth. Average load per minute served as an overload indicator, with values exceeding core count signalling saturation. Results in Fig.37 show CPU load rising with frequency: 10–100 Hz remained within capacity, while higher rates surpassed the four-core threshold. These findings highlight 500 Hz as a practical upper limit for maintaining efficiency without overloading the processor.

The average CPU usage across four cores, and the overall average at varying command rates are shown in Table 12.

Network throughput in Table 13 shows a modest rise in sent and received traffic with increasing command rates, though not strictly linear. At 10 Hz, traffic is 0.011 MB; at 500 Hz, transmission peaks at 0.063 MB and reception at 0.017 MB, then slightly declines at 1000 Hz, suggesting protocol or buffering effects. Overall, network load remains minimal compared to CPU and memory demands, indicat-

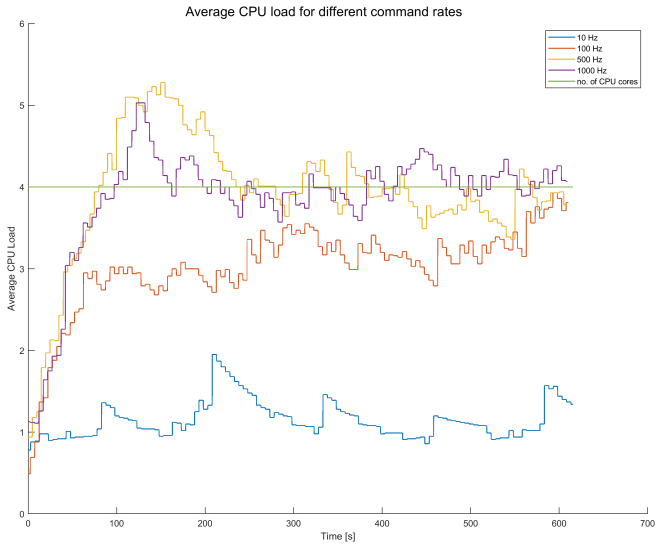


Figure 37: Average CPU Load at Different Command Update Rates

Table 12: Average CPU Load in Total and Per Core at Different Command Update Rates

Update rate [Hz]	Avg. CPU load [%]	CPU1 [%]	CPU2 [%]	CPU3 [%]	CPU4 [%]
10	28.595	27.382	29.638	28.880	27.594
100	65.410	65.628	65.334	64.846	64.635
500	71.690	71.860	71.888	70.347	70.606
1000	71.413	73.181	69.871	71.295	69.175

ing communication is not a performance bottleneck. Memory usage varies only slightly (327–333 MB) across rates, confirming the workload is compute-bound rather than memory-intensive. CPU performance thus dominates system scalability under test conditions.

6.2.1 Discussion on the results

As the command rate increases, CPU usage rises, with the average load going from 28.6% at 10 Hz to over 71% at 1000 Hz. The per-core values remain closely aligned, indicating balanced distribution of workload across cores. Notably, the system reaches peak efficiency around 500 Hz, with minimal variation between cores, while at 1000 Hz, slight divergence suggests the onset of saturation. These results underscore the direct relationship between command frequency and CPU demand, offering

Table 13: Average Network Throughput (Sent and Received) and Memory Used at Different Command Update Rates

Update rate [Hz]	Sent [MB/s]	Received [MB/s]	Memory used [MB]
10	0.011 025	0.010 912	328.67
100	0.047 356	0.013 282	323.83
500	0.062 552	0.016 542	333.20
1000	0.059 135	0.013 771	332.26

insight into system scalability and performance thresholds. The network transmission data shown in Table 13 reveals a modest increase in both sent and received traffic as command rates rise, though the trend is not strictly linear. At 10 Hz, the system transmits and receives approximately 0.011 MB, while at 500 Hz, transmission peaks at 0.063 MB and reception at 0.017 MB. Interestingly, at 1000 Hz, the sent data slightly decreases compared to 500 Hz, and the received data drops back to near 100 Hz levels. This suggests that while higher command rates generally increase network activity, the relationship may be influenced by protocol efficiency, buffering, or saturation effects.

Overall, network usage remains low relative to CPU and memory demands, indicating that communication overhead is not a limiting factor in system performance under these test conditions. The memory usage data in Table 13 shows minimal variation across command rates, with values ranging from 323 MB to 333 MB. Despite the significant increase in CPU load from 10 Hz to 1000 Hz, memory consumption remains relatively stable, suggesting that the workload is primarily compute-bound rather than memory-intensive. The slight up-tick at higher frequencies may reflect increased buffer usage or system overhead, but overall, memory demand does not scale proportionally with command rate. This leads to the conclusion that CPU performance is the dominant factor in system scalability under these test conditions. The data reveals a clear and consistent relationship between command rate and system resource utilization. As the frequency of commands increases from 10 Hz to 1000 Hz, CPU load rises significantly, with average usage approaching the system's core limit at higher rates—indicating potential saturation. Despite this, memory usage remains relatively stable across all rates, suggesting the workload is primarily compute-bound rather than memory-intensive. Network activity shows a modest increase with command rate but remains minimal overall, implying that communication overhead is not a limiting factor.

These findings suggest that while the system scales well up to 500 Hz, pushing beyond this threshold may compromise performance due to CPU constraints, making 500 Hz a practical upper limit for sustained efficiency. Considering that these command frequencies are intended for actuators and motors, the test results offer valuable information on the real-time control capabilities. CPU load scales predictably with frequency, peaking near saturation at 1000 Hz, which could introduce latency or jitter in motor response, which are critical factors in control systems. Given the need for consistent and timely actuation, operating at or below 500 Hz appears optimal, balancing responsiveness with system stability. This threshold ensures reliable motor control without risking performance degradation due to CPU overload. Regarding the sensors, as mentioned previously, they are currently treated as a black-box. However, it's most likely that their physical integration will contribute to an overall increase in system load.

The experimental results show that the software architecture demonstrates reliable and optimal performance for actuator command up to 100 Hz, with balanced CPU usage, stable memory consumption, and minimal network overhead, making it a scalable and efficient choice for real-time motor control.

Overall, the test results show that sustainable rates for commands are 10 Hz and 100 Hz. Both rates are leaving enough resources free to execute other functions, such as higher-level control, perception and supervisory processes, in the AIG platform. Therefore, the proposed architecture is capable of 100 Hz control loops, which aligns with earlier research done in the same context under AIRSHIP project by Tampere University (Hakonen-Milosevic, Aaltonen, & Koskinen, 2025).

7 Conclusion and Future Work

The control strategies developed for the AIRSHIP platform, encompassing algorithmic control, such as cascade PID, Linear Quadratic Gaussian (LQG), Linear Quadratic Regulator with Integral action (LQRI), L1 Adaptive, Fuzzy Control, and AI-based control with model-based reinforcement learning using Dreamer-V3, demonstrated satisfactory performance across a variety of simulated tasks and flight scenarios.

Each approach exhibited distinct advantages. The cascaded PID architecture provided a model-agnostic baseline capable of reliable stabilization and waypoint tracking in ground-effect flight. LQG control demonstrated good disturbance rejection and robustness to measurement noise, while LQRI achieved accurate reference tracking, including during more complex three-dimensional manoeuvres. The $\mathcal{L}_1$ adaptive controller showed strong robustness to parametric uncertainties and unmodeled dynamics, reducing the reliance on gain scheduling. Fuzzy logic control proved particularly effective during the takeoff and lift-off phase, where strong nonlinearities and ground-effect interactions dominate and precise modeling is difficult. RL algorithms demonstrated the potential to learn complex control policies without requiring explicit knowledge of the non-linear aircraft dynamics.

The design and evaluation of these controllers was facilitated by three complementary simulation frameworks: a Python-based simulator, a MATLAB/Simulink environment, and an X-Plane 12 flight simulator. These frameworks allow the combination of aircraft 6DoF dynamics, configurable actuator and sensor models, and support for both classical and learning-based controllers, enabling structured benchmarking across tasks, vehicle iterations, and environmental conditions. The Gym-compatible API for the Python framework makes it suitable for scalable training, analysis, and cross-validation.

Future efforts will focus on extending the current validation and testing pipeline to include additional SIL and HIL experiments, thereby bridging the gap between simulation and real-world deployment. Reinforcement learning controllers, in particular, will be further optimized to improve sample efficiency, robustness to disturbances, and generalization across different flight conditions. The ultimate goal is the deployment of these validated control strategies on the flying prototype, ensuring safe, reliable, and adaptive autonomous flight in ground-effect conditions.

References

Amir, M. A. U., Maimun, A., Mat, S., Saad, M., & Zarim, M. (2016). Wing in ground effect craft: A review of the state of current stability knowledge. In *International conference on ocean mechanical and aerospace for scientists and engineer.*

ArduPilot Development Team. (2009). *ArduPilot.* https://ardupilot.org.

ArduPilot Development Team. (2019). *Mission Planner.* https://ardupilot.org/planner/.

Ayora, M. J. M. (2017). *Guidance and control of a fixed wing aircraft* (Unpublished master's thesis). School of Industrial and Information Engineering, Politecnico di Milano.

Basics of pid control. (2006). London: Springer London. Retrieved from https://doi.org/10.1007/1-84628-586-0_1 doi: 10.1007/1-84628-586-0_1

Beard, R. W., & McLain, T. W. (2012). *Small unmanned aircraft: Theory and practice*. Princeton university press.

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. In *2019 international conference on unmanned aircraft systems (icuas)* (p. 523-533). doi: 10.1109/ICUAS.2019.8798254

Chovet, L. P., Garcia, G. M., Bera, A., Richard, A., Yoshida, K., & Olivares-Mendez, M. A. (2025). Performance comparison of ros2 middlewares for multi-robot mesh networks in planetary exploration. *Journal of Intelligent & Robotic Systems*, *111*(1), 18. Retrieved from https://libkey.io/10.1007/s10846-024-02211-2 doi: 10.1007/s10846-024-02211-2

Dreus, A., Alekseyenko, S., Kulyk, O., & Nekrasov, V. (2025). Prospects for the creation of small-sized high-speed unmanned aerial vehicles based on wig-craft. *EUREKA: Physics and Engineering*(1), 34-43. Retrieved from www.proquest.com%2Fscholarly-journals%2Fprospects-creation-small-sized-high-speed%2Fdocview%2F3211762270%2Fse-2%3Faccountid%3D14242

Dryden, H., & Kuethe, A. (1930). *Effect of turbulence in wind tunnel measurements* (Vol. 342). US Government Printing Office.

Hafner, D., Pasukonis, J., Ba, J., & Lillicrap, T. (2025). Mastering diverse control tasks through world models. *Nature*, *640*, 647–653.

Hakonen-Milosevic, K., Aaltonen, J., & Koskinen, K. (2025, July 3). Practical limitations of in-vehicle ethernet for autonomous mobile machine control. In *2025 international conference on electrical, computer and energy technologies.* Paris, France: IEEE Xplore. (In Press)

Hoisko, A. (2025). *Valokuitu sulautetuissa järjestelmissä, ilmailun sovelluskohteet.* (Finnish)

Iancu, I. (2012). A mamdani type fuzzy logic controller. *Fuzzy logic-controls, concepts, theories and applications*, *15*(2), 325–350.

Ilić, S., Hoisko, A., Milošević, Z., Hakonen-Milosevic, K., Aaltonen, J., Koskinen, K. T., & Domínguez, S. (2025). Control software architecture for enabling autonomous high-speed wing-in-ground vehicles. In *2025 13th international conference on control, mechatronics and automation (iccma 2025).*

Ilić, S., Milošević, Z., Suarez Fernandez, R. A., Rossi, C., & Domínguez, S. (2024). An lqg approach to the control of unmanned wing-in-ground vehicle. In *2024 7th iberian robotics conference (robot)* (p. 1-6). doi: 10.1109/ROBOT61475.2024.10796927

Inc., T. M. (2022). *Matlab version: 9.13.0 (r2022b).* Natick, Massachusetts, United States: Author Retrieved from https://www.mathworks.com

International Maritime Organization. (2018, May). *Guidelines for wing-in-ground craft* (IMO Circular No. MSC.1/Circ.1592). Maritime Safety Committee, International Maritime Organization. (Supersedes MSC/-Circ.1054 (2002) and MSC/Circ.1126 (2004))

ISO 2533:1975 standard atmosphere. (1975). International Organization for Standardization.

Johansen, T. A., & Fossen, T. I. (2020). Guidance, Navigation, and Control of Fixed-Wing Unmanned Aerial Vehicles. In M. H. Ang, O. Khatib, & B. Siciliano (Eds.), *Encyclopedia of Robotics* (pp. 1–9). Berlin, Heidel-

berg: Springer Berlin Heidelberg. Retrieved 2025-11-17, from http://link.springer.com/10.1007/978-3-642-41610-1_67-1 doi: 10.1007/978-3-642-41610-1_67-1

Kai, J.-M., Hamel, T., & Samson, C. (2018). *A unified approach to fixed-wing aircraft path following guidance and control.* arXiv. Retrieved from http://arxiv.org/abs/1803.05184 (arXiv:1803.05184 [cs]) doi: 10.48550/arXiv.1803.05184

Kaplan, E. D., & Hegarty, C. J. (2006). *Understanding GPS: principles and applications* (Second edition ed.). Boston London: Artech House.

Kopečnỳ, L., Hnidka, J., & Bajer, J. (2022). Preliminary design of a fixed-winged uav autopilot for take-off phase and its assessment. In *2022 20th international conference on mechatronics-mechatronika (me)* (pp. 1–6).

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

L. Meier, P. Tanskanen, L. Heng, G. Lee, F. Fraundorfer, and M. Polle feys. (2012). Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, *33*.

Lagarias, J. C., Reeds, J. A., Wright, M. H., & Wright, P. E. (1998). Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM Journal on Optimization*, *9*(1), 112-147. Retrieved from https://doi.org/10.1137/S1052623496303470 doi: 10.1137/S1052623496303470

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

Lekkas, A. M., & Fossen, T. I. (2014). Minimization of cross-track and along-track errors for path tracking of marine underactuated vehicles. In *2014 European Control Conference (ECC)* (pp. 3004–3010). Strasbourg, France: IEEE. Retrieved from http://ieeexplore.ieee.org/document/6862594/ doi: 10.1109/ECC.2014.6862594

Lorenz Meier. (2009). *MAVLink.* https://mavlink.io/en/.

LozioAlce. (2025). *L1_AC: L1 adaptive controller implementation.* https://github.com/LozioAlce/L1_AC.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, *7*(66), eabm6074. Retrieved from https://www.science.org/doi/abs/10.1126/scirobotics.abm6074 doi: 10.1126/scirobotics.abm6074

Mazlan, N. E. A., Shamsudin, S. S., Pairan, M. F., Yaakub, M. F., & Ramli, M. F. (2021). The Design of an Automatic Flight Control System and Dynamic Simulation for Fixed-Wing Unmanned Aerial Vehicle (UAV) using X-Plane and LabVIEW. *Progress in Aerospace and Aviation Technology*, *1*(1). Retrieved from https://publisher.uthm.edu.my/ojs/index.php/paat/article/view/10300 doi: 10.30880/paat.2021.01.01.007

Microsoft flight simulator. (2025). Retrieved from https://www.flightsimulator.com/ (Accessed: 2025-12-23)

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

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

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

Nguyen, A.-T., Taniguchi, T., Eciolaza, L., Campos, V., Palhares, R., & Sugeno, M. (2019). Fuzzy control systems: Past, present and future. *IEEE Computational Intelligence Magazine*, *14*(1), 56-68. doi: 10.1109/MCI.2018.2881644

Nouman, M., & Azam, M. (2023, 04). A systematic review of non-functional requirements mapping into architectural styles. *Bulletin of Electrical Engineering and Informatics*, 1226-1236.

Nugroho, G., Dharmawan, A., Lelono, D., & Handayani, A. M. (2022). *Waypoint Tracking of a Fixed-Wing UAV Using the L1 Cross Track Error Control.* ICIC International ⍰⍰. Retrieved from https://doi.org/10.24507/icicelb.13.02.115 doi: 10.24507/icicelb.13.02.115

Perry, A. R. (2004). The flightgear flight simulator. In *Proceedings of the 2004 usenix annual technical conference.* Retrieved from https://www.usenix.org/legacy/event/usenix04/tech/sigs/full_papers/perry/perry_html/

Phillips, W. F., & Hunsaker, D. F. (2013). Lifting-line predictions for induced drag and lift in ground effect. *31st AIAA Applied Aerodynamics Conference*. Retrieved from https://arc.aiaa.org/doi/abs/10.2514/6.2013-2917 doi: 10.2514/6.2013-2917

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

Robusto, C. C. (1957). The cosine-haversine formula. *The American Mathematical Monthly*, *64*(1), 38–40. Retrieved from http://refhub.elsevier.com/S0038-092X(18)30017-3/h0290

Rugh, W. J., & Shamma, J. S. (2000). Research on gain scheduling. *Automatica*, *36*(10), 1401-1425. Retrieved from https://www.sciencedirect.com/science/article/pii/S0005109800000583 doi: https://doi.org/10.1016/S0005-1098(00)00058-3

Sontag, E. D. (1998). *Mathematical control theory: Deterministic finite-dimensional systems* (Vol. 6). New York, NY, USA: Springer.

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*.

Yalçin, M. K., & Ersoy, E. (2018). Designing autopilot system for fixed-wing flight mode of a tilt-rotor UAV in a virtual environment: X-Plane.

Zadeh, L. A. (1988). Fuzzy logic. *Computer*, *21*(4), 83–93.