POLITECNICO DI TORINO

Master's Degree in COMPUTER ENGINEERING

Politecnico di Torino

EPFL

Master's Thesis

L1-ADAPTIVE AUGMENTATION FOR ROBUST MORPHING-WING DRONE FLIGHT CONTROL

Supervisors:

Julius WANNER *(EPFL)*
Simon JEGER *(EPFL)*
Dario FLOREANO *(EPFL)*
Giuseppe AVERTA *(PoliTO)*

Candidate:

Simone CARLETTI

ACADEMIC YEAR 2025/2026

To all the people I've met over these
years who have made my journey special.

Abstract

Agile flight in cluttered environments presents unique control challenges, particularly for morphing drones subject to complex aerodynamic interactions. While Reinforcement Learning (RL) has emerged as a powerful tool for generating agile control policies, direct actuator control via RL often suffers from a lack of robustness due to model discrepancies, wind disturbances, and ground effects.

This work addresses these limitations by investigating robust control strategies that augment or stabilize RL-based policies. Drawing on techniques proven in quadrotors and fighter aircrafts, we explore the implementation of Model Reference Adaptive Control (MRAC) methods, specifically L1-Adaptive Control (L1AC), to compensate for model discrepancies in real time.

We show that L1 effectively compensates model mismatches, thereby enforcing a consistent dynamic behavior that aligns with the nominal model. We also evaluate how L1 adds to Domain Randomization (DR), specifically investigating whether the combination of offline robust training (via DR) and online adaptation (via L1) outperforms either method individually.

Simulation and experimental results are obtained on morphing drone prototypes developed at EPFL's Laboratory of Intelligent Systems (LIS), demonstrating the effectiveness of the proposed approach in realistic flight conditions.

Keywords: Adaptive Control, Reinforcement Learning, L1 AC, Sim-to-Real, Morphing Drone

Contents

List of Figures

Chapter 1

Introduction

The field of Unmanned Aerial Vehicle (UAV) control is undergoing a significant paradigm shift. Recent advancements in Artificial Intelligence and Machine Learning have driven a significant shift toward learning-based control methods, most notably **Reinforcement Learning (RL)**.

While RL offers the versatility to train agile and complex control policies in simulation, its primary challenge remains the **sim-to-real gap**. This discrepancy arises from inevitable mismatches between the training environment and the physical world, including unmodeled aerodynamic effects, parameter mismatches, and environmental disturbances. In the context of agile flight, where precise tracking and rapid disturbance rejection are critical, these uncertainties can severely degrade performance or lead to catastrophic failure.

To address this challenge, techniques such as **Domain Randomization (DR)** have been explored. DR attempts to robustify policies by training them across a range of randomized physical parameters. Yet, reliance on DR alone often proves insufficient, especially when the model used for training is not exact. Furthermore, excessive randomization can lead to overly conservative policies that sacrifice agility for stability. Consequently, there is a compelling need for low-level adaptive controllers capable of bridging this gap by estimating and compensating for unmatched uncertainties in real-time.

This thesis proposes a hybrid control architecture that integrates $\mathcal{L}_1$ **Adaptive**

Control with a high-level RL-based policy. The primary objective is to develop a control solution that retains the agility of learning-based methods while ensuring robustness against significant model uncertainties and external disturbances.

It is important to underline that, unlike standard multirotor platforms (which benefit from extensive research and established control paradigms) **morphing-wing drones present a significantly higher level of complexity**. Their limited and slower actuators impose severe dynamic constraints, while the morphing capability itself results in highly nonlinear and time-varying dynamics that are difficult to model accurately.

1.1 Contributions

This thesis tries to implement an **end-to-end RL policy** on a morphing drone, with a focus on having a robust real-life deployment. Because the platform's highly constrained, non-instantaneous actuators prevented a direct application of the theory, several critical challenges had to be addressed to ensure reliable flight.

The primary contributions of this work are:

- **End-to-End RL policy augmented with $\mathcal{L}_1$ Adaptive Control:** Development and real-world deployment of an end-to-end RL policy, without any extra middle control layers. Managed to augment this agent with an adaptive controller to reject uncertainties and help bridging the sim-to-real gap.
- **Constrained QP Allocation with Online Matrix Estimation:** Formulation of a control allocation scheme based on Quadratic Programming that handles the highly limited actuation and strict saturation limits. This includes the online computation of the control effectiveness matrix (B), enabling the system to adapt to the vehicle's changing configuration in real time.
- **Transport Delay Compensation:** Enhancement of the $\mathcal{L}_1$ adaptive loop by compensating for communication delays present in our system.

1.2 Thesis Outline

The remainder of this document is structured as follows:

- **Chapter 2** reviews the relevant literature on control strategies for agile flight, the application of Reinforcement Learning, and the foundations of Adaptive Control methodologies.
- **Chapter 3** details the methodology, including the dynamic modeling of the morphing drone, the formulation of the RL policy, and the mathematical derivation of the $\mathcal{L}_1$ Adaptive Controller.
- **Chapter 4** presents the experimental setup, detailing the software and hardware infrastructure, and analyzes the results from both simulation and real-world flight tests.
- **Chapter 5** discusses the implications of the experimental results, highlighting the performance gains and limitations alongside current hardware.
- **Chapter 6** concludes the work and outlines potential ideas for future research
- The **Appendix** contains some code examples from our setup

Chapter 2

Literature Review

The pursuit of autonomous agile flight in Unmanned Aerial Vehicles (UAVs) requires control strategies capable of handling highly nonlinear dynamics and environmental disturbances. This chapter reviews the evolution of flight control methods, shifting from classical model-based approaches to modern learning-based techniques. Furthermore, it analyzes the theoretical foundations of Adaptive Control, which form the core of the robust architecture proposed in this thesis.

2.1 Control Strategies for Agile Flight

Model-based control strategies have long served as the cornerstone of flight controller design. Among these, Model Predictive Control (MPC) has been widely adopted due to its ability to handle multi-variable systems explicitly. MPC operates by optimizing a sequence of control inputs over a finite time horizon, minimizing a cost function while strictly adhering to system constraints.

While highly effective in structured environments, MPC faces significant limitations in very complex scenarios, like agile flight of morphing drones. First it's computationally intensive, requiring the online solution of a convex optimization problem at every time step, which strains onboard embedded hardware. Second, to achieve complex tasks, such as acrobatic maneuvers, we must manually craft a cost function and constraints that describe the desired behavior. In many agile scenarios, the

optimal trajectory is not known a priori, making it difficult to formulate a tractable MPC problem [1].

To overcome these limitations, this work utilizes **Reinforcement Learning (RL)**. RL shifts the computational burden to the offline training phase, yielding a policy that is computationally lightweight for onboard execution. Furthermore, using a learning-based approach we can theoretically achieve any physically feasible maneuver; the primary requirement is simply to design an appropriate reward function and ensure sufficient training.

2.2 Reinforcement Learning and the Sim-to-Real Gap

In recent years, Reinforcement Learning (RL) has enabled autonomous drones to achieve superhuman performance in agile maneuvers, most notably in the domain of competitive drone racing as demonstrated by Scaramuzza's RPG lab [1][2] (fig. 2.1).

However, the transition from simulation to the physical world, the "Sim-to-Real" gap, remains the primary bottleneck for RL deployment. Discrepancies between the physics engine and reality (e.g., sensor noise, communication latency, model uncertainties, etc.) can cause policies trained in simulation to fail on real hardware.

To mitigate this, *Domain Randomization* (DR) is commonly employed [3]. DR randomizes physical parameters (mass, friction, etc.) during training to force the agent to learn robust features. While effective, standard DR has limitations:

1. **Conservatism:** To handle the worst case scenarios, the agent may learn conservative behaviors, sacrificing performances.
2. **Unmatched Dynamics:** DR cannot easily account for structural model mismatches, such as unmodeled dynamics or hardware failures.

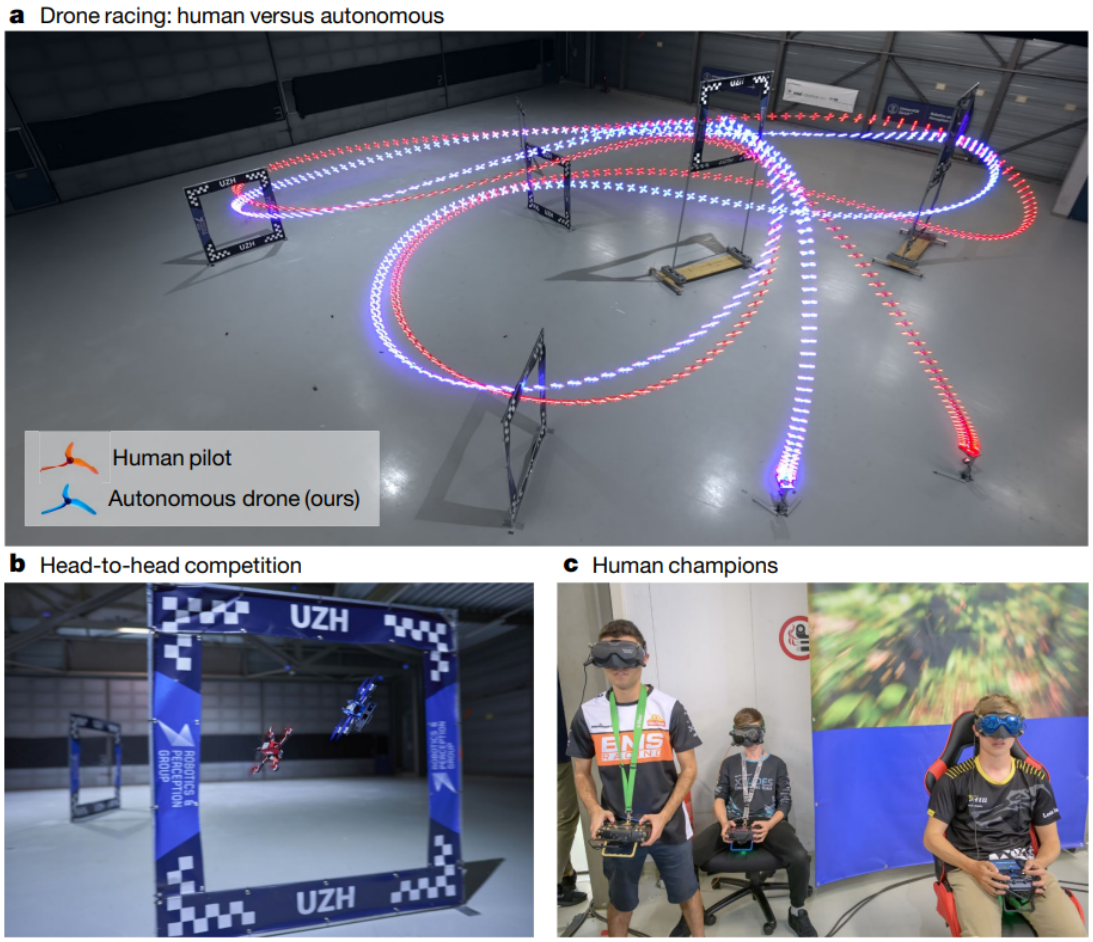


Figure 2.1: Drone racing with Reinforcement Learning [1] - UZH RPG

2.3 Adaptive Control Theory

To bridge the gap left by imperfect models and simulation gaps, adaptive control offers a theoretical framework for estimating and canceling uncertainties online.

2.3.1 Model Reference Adaptive Control (MRAC)

Traditional adaptive control, specifically Model Reference Adaptive Control (MRAC), utilizes a reference model that defines the desired behavior of the system. An adaptive law estimates the uncertainties in real-time and adjusts the control input to force the actual system to track the reference model [4].

While MRAC ensures asymptotic tracking, it suffers from a fundamental trade-off between adaptation speed and robustness. To reject fast-changing disturbances (such as wind gusts), high adaptation gains are required. In practice, high gains often introduce high-frequency oscillations in the control signal, which can excite unmodeled structural dynamics and lead to instability.

2.3.2 $\mathcal{L}_1$ Adaptive Control

The $\mathcal{L}_1$ Adaptive Control ($\mathcal{L}_1$-AC) architecture, proposed by Hovakimyan and Cao [5], was developed specifically to overcome the limitations of MRAC.

The central innovation of $\mathcal{L}_1$-AC is the **decoupling of the adaptation loop from the control loop**. This is achieved by introducing a low-pass filter. The filter limits the bandwidth of the control signal, ensuring that the high-frequency content of the fast adaptation estimates is not passed to the actuators. This architecture allows for arbitrarily fast adaptation rates, ensuring precise estimation of uncertainties without sacrificing robustness. The architecture has shown promising results for agile flight scenarios, as shown in a variety of papers [6], [7], [8], [9], [10], [11], [12] (fig. 2.2, 2.3).

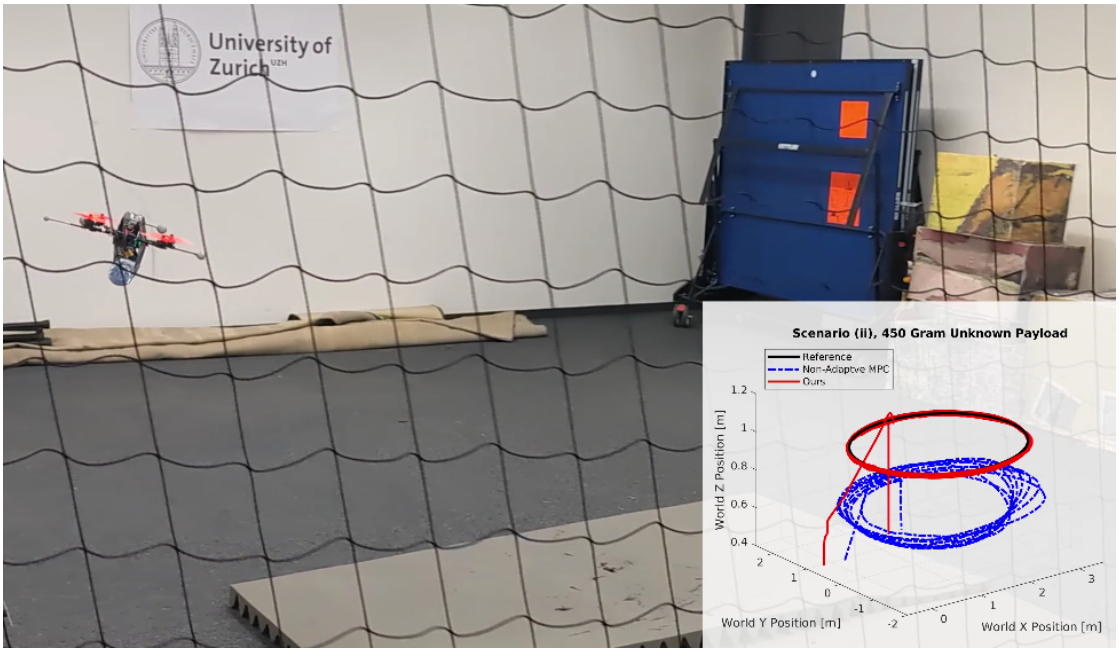


Figure 2.2: MPC + $\mathcal{L}_1$ with a mass offset (beer can) [6]

2.3.3 Control Challenges of Morphing UAVs

While rigid-body multirotors and standard fixed-wing aircraft dominate the literature, avian-inspired morphing drones represent a paradigm shift in UAV design. By dynamically altering their morphology (such as sweeping their wings) these vehicles can seamlessly transition between high-efficiency cruising and high-agility maneuvering. However, this physical adaptability introduces complex control challenges.

The primary difficulty lies in the highly nonlinear and time-varying nature of the morphing dynamics. A change in wing sweep directly alters the aircraft's wingspan, active lifting surface area, and inertial tensor. To address these challenges, recent literature has explored various nonlinear and adaptive control strategies, such as (Nonlinear Incremental) Dynamic Inversion and $\mathcal{L}_1$ Adaptive Control [10]. Within this, significant advancements have been also made by the Laboratory of Intelligent Systems (LIS) at EPFL. Recent works from the laboratory have successfully demonstrated the potential and agile flight capabilities of these avian-inspired drones [13], [14] (fig. 2.4).

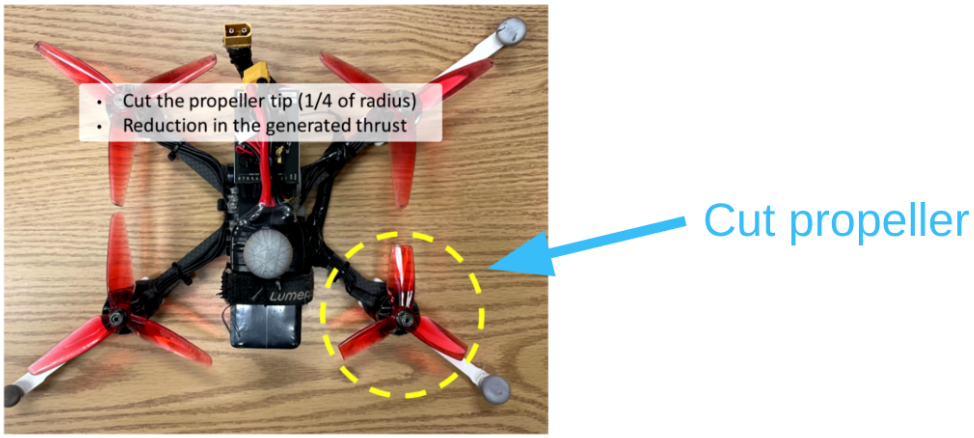


Figure 2.3: In [12] $\mathcal{L}_1$-AC successfully corrected a cut propeller on a quadrotor

While these foundational works successfully navigate the mechanics of morphing flight, achieving robust and agile flight in unpredictable real-world environments remains an open challenge. Classical and model-based approaches inherently rely on accurate analytical models. This fundamental reliance on precise, difficult-to-obtain aerodynamic modeling strongly motivates the shift toward learning-based control policies, provided the resulting sim-to-real gap can be robustly bridged.

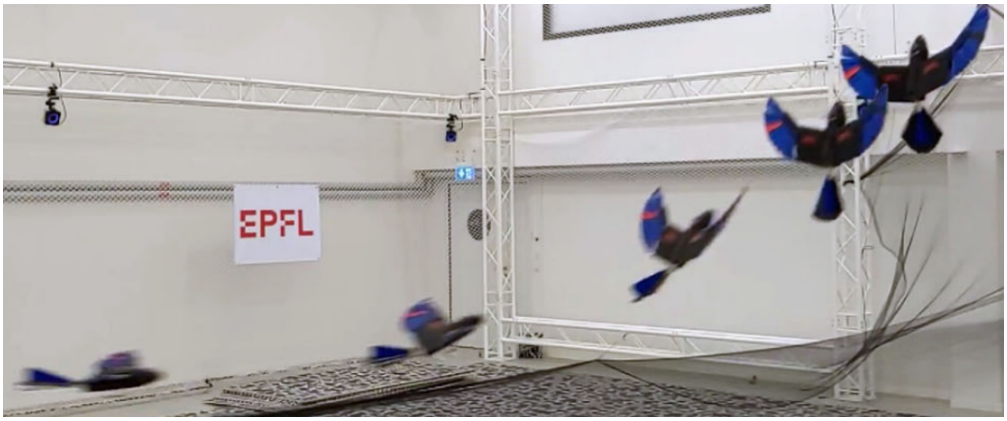


Figure 2.4: Agile flight of the *LisEagle* [14]

2.4 Summary

The literature suggests that while RL provides a capability for generating agile trajectories, it lacks inherent robustness against model discrepancies. This thesis proposes a hierarchical approach combining a high-level RL policy with low-level $\mathcal{L}_1$-AC to guarantee both agility and reliability in flight. While this approach has yielded promising results for a simpler pendulum system [15], deploying such a framework end-to-end on a highly constrained morphing UAV represents a novel and significant challenge, which is addressed in the following chapters.

Chapter 3

Methodology

This chapter details the design and implementation of the proposed robust control architecture. Section 3.1 introduces the morphing drone platform and its dynamic modeling. Section 3.2 describes the formulation of the Reinforcement Learning (RL) policy, including the observation space and reward structure. Finally, Section 3.3 present the derivation and integration of the $\mathcal{L}_1$ Adaptive Controller.

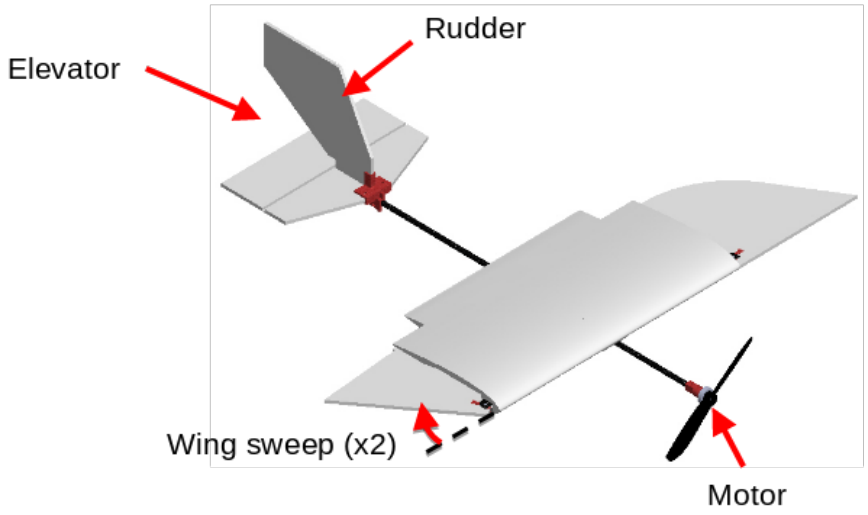
3.1 System Overview and Modeling

The experimental platform utilized in this work is a custom **avian-inspired morphing drone** developed by the Laboratory of Intelligent Systems (LIS) of EPFL. Unlike standard rigid-frame drones, this platform features **morphing wings** which allows it to alter its aerodynamic profile in flight.

3.1.1 Dynamics Model

The UAV is modeled as a 6-Degree-of-Freedom (6-DOF) rigid body. We define the world frame $\mathcal{F}_W$ using the North-West-Up (NWU) convention and the body frame $\mathcal{F}_B$ using the Front-Left-Up (FLU) convention. The full system state $\mathbf{x} \in \mathbb{R}^{13}$ is defined as:

$$\mathbf{x} = \left[\mathbf{p}^\top, \mathbf{q}^\top, \mathbf{v}^\top, \boldsymbol{\omega}^\top\right]^\top \tag{3.1}$$



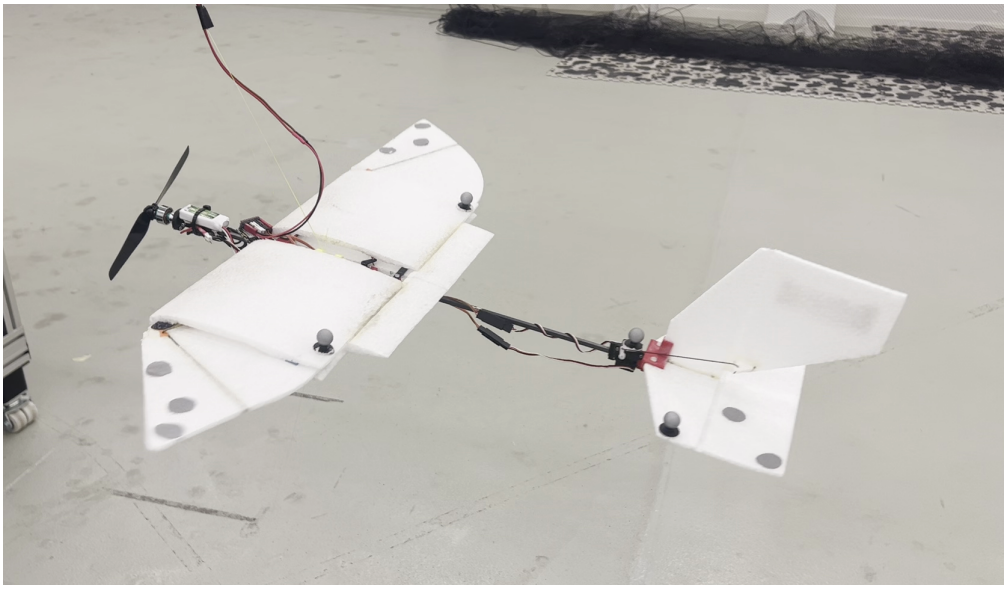


Figure 3.1: The drone (aka *Lisparrow*)

where:

- $\mathbf{p} \in \mathbb{R}^3$ is the position in the world frame $\mathcal{F}_W$.
- $\mathbf{q} \in \mathbb{R}^4$ is the unit quaternion representing the attitude of $\mathcal{F}_B$ relative to $\mathcal{F}_W$.
- $\mathbf{v} \in \mathbb{R}^3$ is the linear velocity expressed in the **body frame** $\mathcal{F}_B$.
- $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular velocity expressed in the **body frame** $\mathcal{F}_B$.

For control design, we utilize a reduced state vector containing only the body-frame velocities:

$$\mathbf{x}_{vw} = \left[\mathbf{v}^\top, \boldsymbol{\omega}^\top\right]^\top \tag{3.2}$$

In subsequent sections, this reduced state may be denoted simply as x for brevity. The control input vector $\mathbf{u} \in \mathbb{R}^5$ is comprised of the following:

$$\mathbf{u} = [T, \mathrm{sw}_L, \mathrm{sw}_R, \mathrm{ele}, \mathrm{rud}]^\top \tag{3.3}$$

The actuators are subject to physical saturation limits defined by the hardware in use. The units and feasible ranges for each control input are summarized in Table 3.1. Compared to traditional winged drones our drone implements full wing sweep, which allows us to changes the wing's surface area span. This effectively modifies the "gain" of the aerodynamic forces: a swept-back wing generates different lift/drag magnitudes than a fully extended wing.

It's important to underline that the entire set of actuators features both **limited operational ranges and non-negligible actuator dynamics**, making the platform significantly more constrained than a standard quadrotor. This inherent limitation imposes specific control restrictions that are detailed and demonstrated in the following sections.

Table 3.1: Actuator Limits and Units

Description	**Symbol**	**Unit**	**Range [Min, Max]**
Thrust Magnitude	T	N	$[0.0, 0.85]$
Wing Sweep	$\mathrm{sw}_L, \mathrm{sw}_R$	deg	$[-5.0, 75.0]$
Elevator	ele	deg	$[-26.0, 26.0]$
Rudder	rud	deg	$[-30.0, 30.0]$

3.1.2 Rigid Body Dynamics

The time evolution of the system is governed by the Newton-Euler equations of motion:

$$\dot{\mathbf{p}} = \mathbf{R}_{WB}(\mathbf{q})\mathbf{v} \tag{3.4}$$

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \tag{3.5}$$

$$\dot{\mathbf{v}} = \frac{1}{m}\mathbf{F}_{\text{ext}} - \boldsymbol{\omega} \times \mathbf{v} \tag{3.6}$$

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}\left(\mathbf{M}_{\text{ext}} - \boldsymbol{\omega} \times (\mathbf{J}\boldsymbol{\omega})\right) \tag{3.7}$$

where m is the total mass, $\mathbf{J}$ is the inertia tensor, $\mathbf{R}_{WB}(\mathbf{q})$ is the rotation matrix from the body frame to the world frame, and $\otimes$ denotes quaternion multiplication.

3.1.3 External Forces and Moments

The total external force $\mathbf{F}_{\text{ext}}$ and moment $\mathbf{M}_{\text{ext}}$ acting on the center of gravity (CG) are composed of gravitational, propulsive, and aerodynamic contributions:

$$\mathbf{F}_{\text{ext}} = \mathbf{F}_g + \mathbf{F}_{\text{prop}} + \sum_i \mathbf{F}_{\text{aero},i} \tag{3.8}$$

$$\mathbf{M}_{\text{ext}} = (\mathbf{p}_{\text{prop}} \times \mathbf{F}_{\text{prop}}) + \sum_i (\mathbf{p}_{\text{cp},i} \times \mathbf{F}_{\text{aero},i}) \tag{3.9}$$

where $i \in \{\text{wing}_L, \text{wing}_R, \text{elevator}, \text{rudder}\}$ represents the aerodynamic surfaces, $\mathbf{p}_{\text{prop}}$ is the propeller position, and $\mathbf{p}_{\text{cp},i}$ is the center of pressure for the i-th surface relative to the CG.

The propulsive force $\mathbf{F}_{\text{prop}}$ is generated by the thruster aligned with the body x-axis, while the gravitational force $\mathbf{F}_g$ is defined in the inertial frame and rotated into the body frame using the inverse orientation $\mathbf{R}_{WB}(\mathbf{q})^\top$:

$$\mathbf{F}_{\text{prop}} = \begin{bmatrix} T & 0 & 0 \end{bmatrix}^T, \quad \mathbf{F}_g = \mathbf{R}(\mathbf{q})^\top \begin{bmatrix} 0 & 0 & mg_z \end{bmatrix}^T \tag{3.10}$$

where T is the scalar thrust magnitude and $g_z = -9.81\,\mathrm{m/s^2}$ is the gravitational acceleration along the inertial z-axis.

3.1.4 Actuation and Aerodynamics

Finally, we derive the forces and moments generated by the control surfaces (wings and tail). Since the real actuators are not instantaneous, in order to ensure realistic simulation fidelity and robust control performance, we need to account for the dynamics, delays, and saturation limits. For this reason the generation of the forces follows a cascaded process: $\mathbf{u} \rightarrow$ actuator dynamics $\rightarrow \mathbf{F}, \mathbf{M}$.
The high-level control commands $\mathbf{u}$ first pass through dynamic actuator models to determine the real physical state of the drone (surface deflections and rotor speed), then these physical states pass through the standard rigid-body dynamics to obtain the evolution of the system.

The servos for the wings and tail are modeled as LTI systems (2^{nd} order for the sweep and 1^{nd} order for everything else). For all of them, we also model transport delay using a simple circular buffer, where the delay value is estimated through experimental measurements.

3.2 High-Level Controller: Reinforcement Learning

As mentioned before, this work uses a RL policy to plan and control the drone in various settings. We formulate the control problem as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ represents the transition dynamics, $\mathcal{R}$ is the reward function, and γ is the discount factor.

The high-level policy $\pi_\theta(a_t|o_t)$ is trained using Proximal Policy Optimization (PPO) [16], a state-of-the-art on-policy gradient method. It's objective is to maximize the expected sum of discounted rewards:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T-1} \gamma^t\, r(s_t, a_t)\right] \tag{3.11}$$

3.2.1 Observation Space

The observation space $\mathcal{S} \in \mathbb{R}^{18}$ is designed to provide the agent with full knowledge of its kinematic state while remaining independent of absolute position and specific trajectories. The observation vector o_t at time step t is defined as:

$$o_t = \left[\mathbf{p}_{g,t}^\top, \mathrm{vec}(\mathbf{R}_t)^\top, \mathbf{v}_t^\top, \boldsymbol{\omega}_t^\top\right]^\top \tag{3.12}$$

where all quantities are expressed in the **body frame**:

- $\mathbf{p}_{g,t} \in \mathbb{R}^3$ is the relative position vector pointing to the next target waypoint.
- $\mathbf{R}_t \in SO(3)$ is the attitude rotation matrix. It is flattened into a vector $\mathrm{vec}(\mathbf{R}_t) \in \mathbb{R}^9$.
- $\mathbf{v}_t \in \mathbb{R}^3$ is the linear velocity.
- $\boldsymbol{\omega}_t \in \mathbb{R}^3$ is the angular velocity.

3.2.2 Action Space

The action space $\mathcal{A} \subset \mathbb{R}^5$ corresponds to the normalized control inputs. The agent outputs a vector $a_t \in [-1, 1]^5$, which is scaled to the physical actuator limits (see Table 3.1):

$$a_t = [\hat{u}_1, \hat{u}_2, \cdots, \hat{u}_5]^\top \tag{3.13}$$

Each component $\hat{u} \in a_t$ relates to its corresponding physical actuator command u via the normalization:

$$\hat{u} = \frac{u - \mu}{\sigma} \qquad \mu = \frac{u_{\max} + u_{\min}}{2},\ \sigma = \frac{u_{\max} - u_{\min}}{2} \tag{3.14}$$

3.2.3 Reward Function

The reward function r_t plays a central role in shaping the behavior of the RL agent. We use a composite reward that encourages accurate trajectory tracking while also promoting smooth control and energy efficiency. At each time step t, the total

reward is computed as a sum of four components:

$$r_t = r_{\text{traj}} + r_{\text{stab}} + r_{\text{eff}} + r_{\text{term}} \tag{3.15}$$

Trajectory Tracking (r_{traj})

For simple **stabilization** tasks (e.g., hovering), the reward is defined as the weighted Euclidean distance to the goal position $\mathbf{g}$:

$$r_{\text{traj}} = -\lambda_{\text{pos}} \|\mathbf{p}_t - \mathbf{goal}\| \tag{3.16}$$

In contrast, for the task of **trajectory** tracking, we adopt a progress-based formulation based on autonomous drone racing literature [17]: the reward is split into *progress* along the track and *cross-track error*.
We calculate the scalar progress s_t by projecting the drone's position onto the path segment vector connecting the previous waypoint $\mathbf{g}_k$ to the target waypoint $\mathbf{g}_{k,\text{next}}$:

$$s_k = (\mathbf{p}_k - \mathbf{g}_k) \frac{\mathbf{g}_{k,\text{next}} - \mathbf{g}_k}{\|\mathbf{g}_{k,\text{next}} - \mathbf{g}_k\|} \tag{3.17}$$

The total reward is given by:

$$r_{\text{traj}} = -\lambda_{\text{prog}}(s_t - s_{t-1}) - \lambda_{\text{pos}} \frac{\ln(1 + \alpha d_t)}{\alpha} - \lambda_{\text{waypoint}} \mathbb{1}_{\text{waypoint}} \tag{3.18}$$

where:

- The first term rewards the incremental advancement along the trajectory.
- d_t is the perpendicular distance to the path (cross-track error), which is penalized via a log-scaling function parameterized by $\alpha = 0.1$.
- $\mathbb{1}_{\text{waypoint}}$ is a sparse bonus reward given when the agent successfully passes a waypoint ($k \to k+1$), currently parametrized with $\lambda_{\text{waypoint}} = 10\lambda_{\text{prog}}$.

Stability and Regularization (r_{stab})

To prevent oscillatory behavior and ensure the generated commands are physically feasible, we penalize velocities and accelerations. Additionally, orientation penalties

λ_{ori}, λ_{yaw} are applied to encourage the drone to maintain a stable desired attitude and to avoid skewed flights.

$$r_{\text{stab}} = -\lambda_v\|\mathbf{v}_t\| - \lambda_\omega\|\boldsymbol{\omega}_t\|^2 - \lambda_a\|\mathbf{a}_t\| - \lambda_\alpha\|\boldsymbol{\alpha}_t\| \tag{3.19}$$

$$- \lambda_{ori}\|\mathbf{q}_t \ominus \mathbf{q}_{ref}\| - \lambda_{yaw}|\text{yaw}(\mathbf{q}_t)| \tag{3.20}$$

Efficiency and Smoothness (r_{eff})

To again ensure the generated commands are physically feasible but also optimal, we can penalize the energy and the rate of change of the action/control:

$$r_{\text{eff}} = -\lambda_{\text{energy}}\|\mathbf{u}_t\| - \lambda_{\Delta u}\|\mathbf{u}_t - \mathbf{u}_{t-1}\| \tag{3.21}$$

Termination (r_{term})

Finally, a terminal reward r_{term} is assigned if the episode reaches a terminal state. This occurs if the agent exits the bounding box or if the simulation dynamics generate infeasible accelerations:

$$r_{\text{term}} = -\lambda_{\text{term}}\,\mathbb{1}_{\text{terminal}} \tag{3.22}$$

3.2.4 Domain Randomization

To facilitate Sim-to-Real transfer, we apply Domain Randomization at the start of each training episode. This technique introduces variability into the simulation, preventing the agent from overfitting to the nominal dynamics and forcing it to learn an adaptive control strategy.

In our case we add randomization to the initial state x_0, to the observations o_t, and to the physical parameters of the drone (including mass, inertia, geometric features, aerodynamic properties, and actuator delays and parameters). For all of these, the intensity of the randomization is defined by various configurable parameters, denoted here as λ_{DR} for brevity. For a given nominal parameter ϕ_{nom},

the randomized value ϕ_{rand} is sampled from a uniform distribution:

$$\phi_{\text{rand}} = \phi_{\text{nom}} \left(1 + \lambda_{\text{DR}} \delta\right), \quad \text{where} \quad \delta \sim \mathcal{U}(-1, 1) \tag{3.23}$$

This formulation ensures that parameters vary continuously within a bounded range. For instance, a setting of $\lambda_{\text{DR}} = 0.1$ subjects the system to variations anywhere within $\pm 10\%$ of the nominal value, rather than testing only the extreme bounds.

3.3 Low-Level Controller: $\mathcal{L}_1$ Adaptive Control

While the RL agent provides agility, it lacks guarantees against unmodeled disturbances. To address this, we augment the loop with $\mathcal{L}_1$ Adaptive Control ($\mathcal{L}_1$-AC).

The proposed architecture is illustrated in Fig. 3.2. The $\mathcal{L}_1$-AC layer augments the nominal control input from the RL agent to compensate for dynamic uncertainties and external disturbances. Importantly, the RL agent is trained only on the nominal model without this augmentation (see Fig. 3.3). The $\mathcal{L}_1$ controller is introduced only during the deployment phase to robustify the pre-trained policy.

To effectively achieve this, the system operates in a multi-rate configuration. The high-level RL policy computes the nominal control commands at 25 Hz, while the low-level $\mathcal{L}_1$ adaptive loop runs at a significantly faster rate of 200 Hz. Running the adaptive loop as fast as the hardware permits is a fundamental design requirement of the $\mathcal{L}_1$ architecture. A high sampling rate allows the adaptation law to rapidly capture disturbances and errors, while the low-pass filter ensures that only smooth, bounded frequencies are passed to the actuators. The main components of this low-level controller are described below.

3.3.1 State Predictor

The state predictor estimates the expected evolution of the system, i.e. how the real system should behave under the nominal dynamics. Its purpose is to provide an internal model of the system that is used to estimate the lumped uncertainties affecting the dynamics.

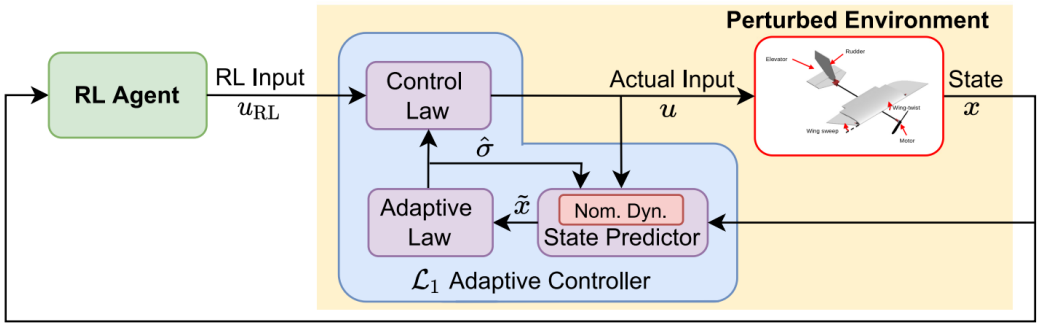


Figure 3.2: The control architecture [15]

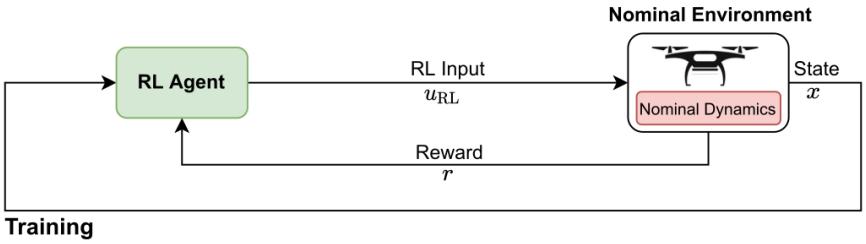


Figure 3.3: The training setup [15]

The predictor is defined as:

$$\dot{\hat{x}}(t) = f(x(t), u(t)) + \hat{\sigma}(t) + A_s\tilde{x}(t) \tag{3.24}$$

where $\hat{x}(t)$ is the predicted state, $f(x, u)$ represents the nominal system dynamics, and $\hat{\sigma}(t)$ is the estimate of the lumped disturbance affecting the system. The term

$$\tilde{x}(t) \triangleq \hat{x}(t) - x(t) \tag{3.25}$$

denotes the state prediction error, while A_s is a diagonal matrix with non-positive entries (this is one of the tunable parameters of the controllers).
For practical implementation, the predictor is discretized using a forward Euler integration scheme. Denoting by k the discrete-time index and by Δt the sampling period, the discrete-time predictor becomes:

$$\hat{x}_{k+1} = \hat{x}_k + \dot{\hat{x}}_k \, \Delta t \tag{3.26}$$

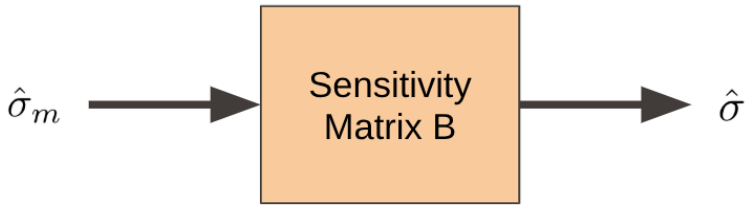


Figure 3.4: Mapping between $\hat{\sigma}_m$ and $\hat{\sigma}$

3.3.2 Adaptation Law

The adaptation law estimates the generalized unmodeled disturbance, denoted by $\hat{\sigma}_k$, using the state prediction error. For practical implementation, we adopt a piecewise-constant (PWC) discrete adaptive law, given by:

$$\hat{\sigma}_k = \Gamma\, \tilde{x}_k \qquad , \qquad \Gamma \triangleq -\Big(e^{A_s \Delta t} - I\Big)^{-1} A_s e^{A_s \Delta t} \tag{3.27}$$

where again $\tilde{x}_k = \hat{x}_k - x_k$ is the state prediction error and Δt is the sampling time.

In standard $\mathcal{L}_1$ adaptive control implementations, the generalized disturbance estimate $\hat{\sigma}$ is typically mapped to actuator-level commands $\hat{\sigma}_m$ (*matched* disturbance) using a pseudo-inverse of the sensitivity matrix, $B^{\dagger}$. However, this approach does not explicitly account for actuator dynamics and limits, which can lead to infeasible control commands in our systems.

Control Allocation via QP

To address these limitations, we use a constrained control allocation scheme based on Quadratic Programming (QP). The matrix $B \in \mathbb{R}^{6\times n_{act}}$ is computed online using a finite-difference approximation (because of the slow nature of our actuators, we can update it only at a frequency of 10 Hz):

$$B_{:,i} \approx \frac{f(x_k,\ \mathrm{sat}(u_k + \varepsilon_i e_i)) - f(x_k,\ \mathrm{sat}(u_k - \varepsilon_i e_i))}{\varepsilon_i^{tot}} \tag{3.28}$$

where e_i is the i-th canonical basis vector, ε_i is a small perturbation applied to the i-th actuator input, and $\text{sat}(\cdot)$ denotes the actuator saturation function that clamps the command to the admissible actuator limits. The term ε_i^{tot} represents the effective perturbation after saturation:

$$\varepsilon_i^{tot} = \text{sat}(u_{k,i} + \varepsilon_i) - \text{sat}(u_{k,i} - \varepsilon_i). \tag{3.29}$$

At each time step k, the matched uncertainty $\hat{\sigma}_m$ is obtained by solving the following optimization problem:

$$\min_{\hat{\sigma}_m} \ \frac{1}{2}\|B\hat{\sigma}_m - \hat{\sigma}_k\|_2^2 + \mathcal{R}(\hat{\sigma}_m) \tag{3.30}$$

$$\text{s.t. } u_{\min} - u_{RL} \leq \hat{\sigma}_m \leq u_{\max} - u_{RL}, \tag{3.31}$$

The regularization term $\mathcal{R}(\hat{\sigma}_m)$ improves flight stability and contains three components:

$$\mathcal{R}(\hat{\sigma}_m) = \underbrace{\sum_{i=1}^{n_{\text{act}}} \alpha_i(\hat{\sigma}_{m,i} - \hat{\sigma}_{m,i}^{\text{prev}})^2}_{\text{rate / smoothness}} + \underbrace{\sum_{(i,j)\in\mathcal{P}} \beta_{ij}(\hat{\sigma}_{m,i} - \hat{\sigma}_{m,j})^2}_{\text{asymmetry}} + \underbrace{\sum_{i=1}^{n_{\text{act}}} \gamma_i \hat{\sigma}_{m,i}^2}_{\text{energy / magnitude}} \quad . \tag{3.32}$$

The first term penalizes rapid variations in the adaptive control output (weighted by α_i), limiting solutions that result in high-frequency actuator oscillations. The second term penalizes asymmetry between some actuator pairs $\mathcal{P}$ (weighted by β_{ij}), encouraging geometrically consistent actuation (e.g., when symmetric sweep is desired). Finally, the third term penalizes large controls, trying to minimize the overall energy of the adaptive commands.

The resulting quadratic program is solved using the active-set solver `qpOASES`, which allows real-time execution with typical solve times below 1 ms.

3.3.3 Control Law (Low-Pass Filter)

Finally, we apply a low-pass filter to the estimated matched disturbance $\hat{\sigma}_m$. This filtering stage is a **key component** of the $\mathcal{L}_1$ adaptive control architecture: it **decouples the adaptation rate from the control bandwidth**, allowing the

system to estimate uncertainties rapidly while ensuring that only smooth and bounded commands are applied to the actuators.

The adaptive control signal is defined as:

$$u_{AD,k} = -\text{LPF}(\hat{\sigma}_{m,k}) \tag{3.33}$$

For a first-order low-pass filter with cutoff frequency ω, the filter dynamics are discretized using a forward Euler approximation, obtaining the discrete-time implementation:

$$u_{AD,k} = -(\omega \Delta t)\, \hat{\sigma}_{m,k} + (1 - \omega \Delta t)\, u_{AD,k-1}. \tag{3.34}$$

The total control input applied to the drone is obtained by combining the nominal reinforcement learning policy with the adaptive control:

$$u_k = u_{RL,k} + u_{AD,k}. \tag{3.35}$$

3.3.4 Transport Delay Compensation

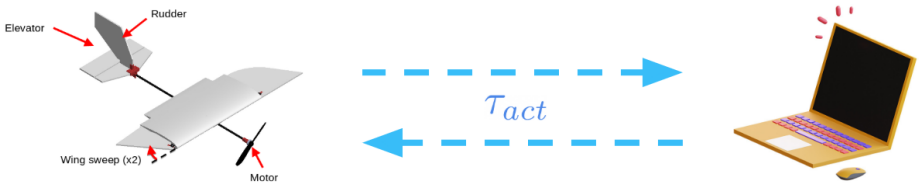


Figure 3.5: Transport delay

In our setup, the control algorithm runs off-board on a laptop, introducing a transport delay τ_{act} due to communication and processing (fig. 3.5). As a result, the state measurement $x(t)$ corresponds to a system state that is older than the state assumed by the controller when computing the adaptive control. This mismatch leads to an incorrect evaluation of the prediction error $\tilde{x}$ and may lead to an unstable controller.

To compensate for this delay, the measured state is propagated forward in time so that it aligns with the controller's time horizon. Given the current measured state $x(t)$ and the history of previously applied control inputs u_{hist}, the delay-

compensated state x_{corr} is obtained by forward integrating the system dynamics:

$$x_{corr}(t) = x(t) + \int_{t}^{t+\tau_{act}} f(x(\tau), u_{hist}(\tau))\, d\tau. \tag{3.36}$$

In the discrete-time implementation, this integral is approximated by re-simulating the system dynamics over the next $N = \lceil \tau_{act}/\Delta t \rceil$ integration steps using a buffer of the most recently applied control inputs. The resulting state estimate x_{corr} is then used in place of the measured state x.

It's important to notice that, while we compensate for this delay, the performance degradation is not completely eliminated, since the projection relies on the same nominal dynamics that are subject to inaccuracies.

Chapter 4

Experimental Results

This chapter evaluates the performance of the proposed control architecture. The primary objective is to demonstrate that augmenting a Reinforcement Learning (RL) policy with an $\mathcal{L}_1$ Adaptive Controller bridges the sim-to-real gap and improves flight performance under uncertainties.

Section 4.1 outlines the software and hardware environment used for testing. Section 4.2 presents results in a hover scenario, while Section 4.3 details the results for an agile flight scenario.

4.1 Experimental Setup

4.1.1 Software and Simulation Framework

The setup builds upon the existing infrastructure utilized by the Laboratory of Intelligent Systems (LIS). The core software stack relies on a customized fork of *Flightmare*, a drone simulator originally developed by the Robotics and Perception Group at the University of Zurich. This specific framework was utilized because it natively integrates Reinforcement Learning (RL) while supporting the addition of custom low-level controllers. Furthermore, it provides a physical and visual simulation environment rendered via Unity (fig. 4.1).

Figure 4.1: Start screen of the Unity simulator

For the purposes of this thesis, this existing lab framework was adapted to accommodate the new setup, adding the new controller and related necessary software interfaces to evaluate the $\mathcal{L}_1$ adaptive augmentation alongside the baseline RL policy.

4.1.2 Hardware and State Estimation

Real-world flight experiments rely on an control pipeline that runs off-board on a laptop. This laptop is then connected via Ethernet (to reduce latency) to a local network. A local router connects to the laptop used for the control and a secondary PC (Figure 4.4b). Real-world state estimation is achieved using an OptiTrack motion capture system (Figure 4.4a) managed by this secondary PC, which is also responsible for controlling a WindShape wind tunnel (Figure 4.4c) utilized during the hovering tests. A comprehensive view of this setup is shown in 4.2.

The laptop runs ROS1 to handle the mocap data streams, system timers, and the main control loops. The OptiTrack system provides high-precision measurements of the drone's position p and attitude q, which are numerically differentiated and filtered to estimate the linear velocity v and angular velocity ω. The computed control commands are then transmitted via UDP to an ESP32 microcontroller

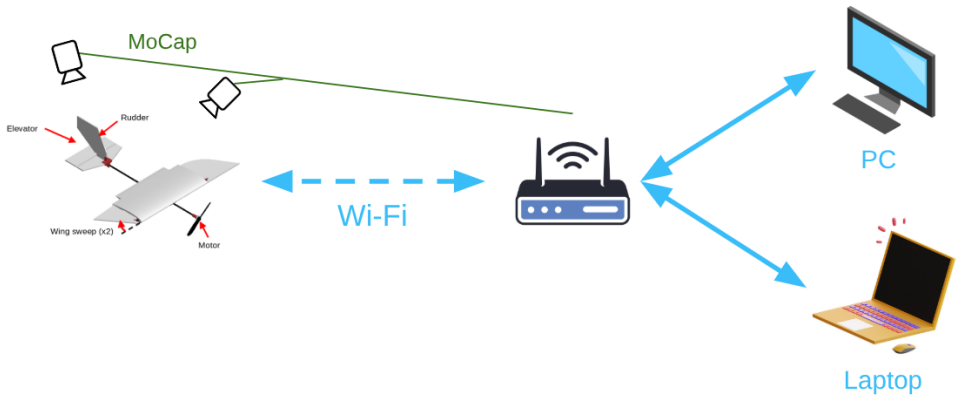


Figure 4.2: Communication setup

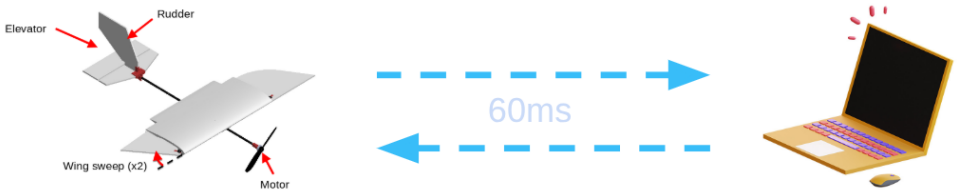


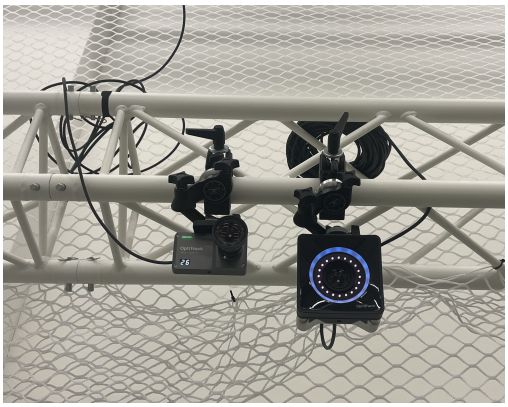
Figure 4.3: Communication Delay

(Figure 4.4d) onboard the drone, which is connected to the local network via Wi-Fi. This whole communication setup introduces a transport delay (τ_{act}) of approximately $60ms$ (fig. 4.3), which is handled by projecting the measured state forward as explained in Section 3.3.4.

4.1.3 Simulation and Test Scenarios

To validate the proposed control architecture, extensive evaluations were conducted in both **simulation** and the **real-world**. The primary objective was to assess the system's ability to maintain stable trajectory tracking under significant uncertainties, specifically mass offset, across two primary flight regimes:

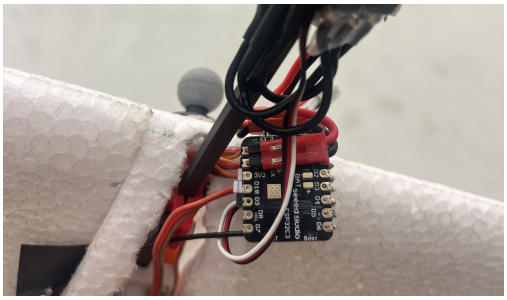
- **Hover Scenario**: Maintaining a fixed setpoint in the presence of a constant 6 m/s wind.



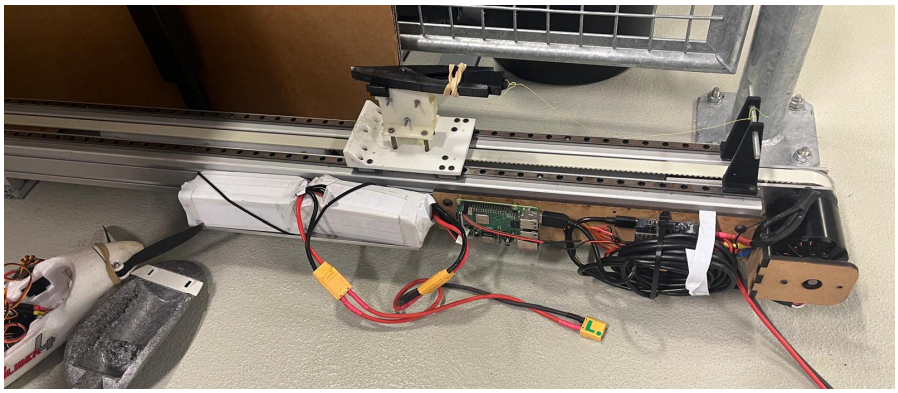
(a) OptiTrack motion capture system.

(b) Secondary PC for mocap and wind tunnel

(c) WindShape wind tunnel for hover tests



(d) Onboard ESP32 microcontroller



(e) Drone launcher for the agile flight

Figure 4.4: Hardware components for real-world flight tests

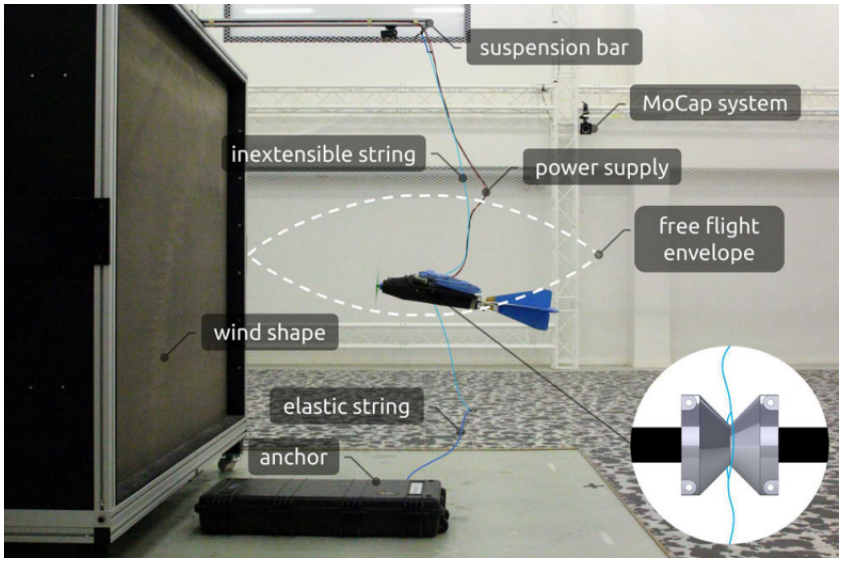


Figure 4.5: Hover test setup (same as [13]).

- **Agile Scenario**: Executing a pitch-up maneuver after a 6 m/s launch.

In simulation, both scenarios were tested using the *Flightmare* simulator (fig. 4.1). Real-world experiments were conducted using dedicated setups for each scenario: the hover experiment was performed in a wind tunnel with a safety tether (fig. 4.5), while the agile scenario used a launcher with a protective safety net (fig. 4.6).

4.2 Hover Scenario

For the **hover** scenario we compared (in simulation) four control configurations to isolate the contributions of DR and $\mathcal{L}_1$-AC:

- **RL (Baseline):** policy trained without DR.
- **RL + DR:** policy trained using DR.
- **RL + $\mathcal{L}_1$:** the baseline RL policy augmented with $\mathcal{L}_1$-AC.
- **RL + DR + $\mathcal{L}_1$:** the complete architecture utilizing both DR and $\mathcal{L}_1$-AC.

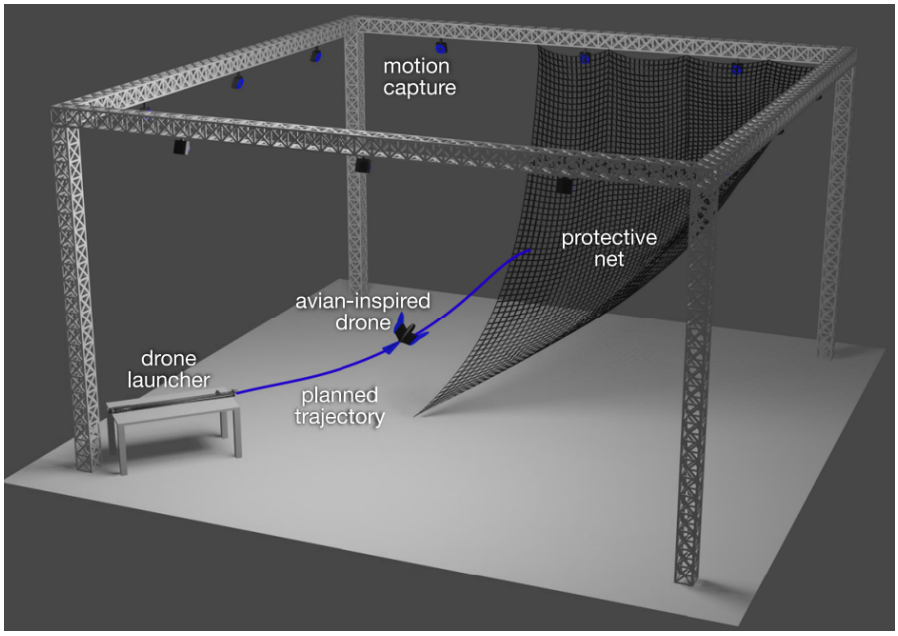


Figure 4.6: Agile test setup (same as [14])

4.2.1 Evaluation Metrics

The performance of each configuration was quantified using two primary metrics:

- **Mean Reward:** This metric reflects the overall task success based on the defined reward function (see (3.15)).
- **Root Mean Square Error (RMSE) to Goal:** This metric measures the tracking error between the drone and the desired hovering setpoint.

4.2.2 Results

The results from the **hover scenario** demonstrate that the combination of both offline (DR) and online ($\mathcal{L}_1$) robustness techniques yields the highest performance. As shown in the comparison of Mean Total Rewards (see fig. 4.8), the baseline RL policy struggled significantly with +50% mass offset, while the **fully augmented** RL + DR + $\mathcal{L}_1$ **configuration achieved the best result**.

Even in cases where DR is already present, the addition of $\mathcal{L}_1$ augmentation

Figure 4.7: Hover in simulation

significantly **stabilizes the control signal**, reducing high-frequency oscillations and ensuring smoother actuator commands during deployment (see fig. 4.9).

Tracking precision reflects this trend, with the combined architecture reaching the lowest RMSE to the goal compared to all other configurations (see fig. 4.10).

Real-world validation in a *WindShape* wind tunnel confirmed these results (see fig. 4.11).

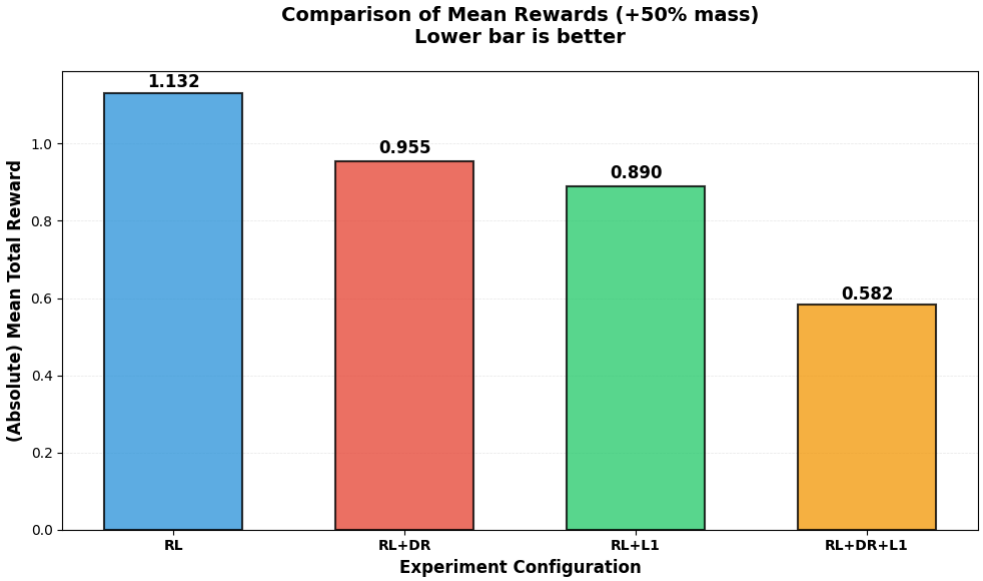


Figure 4.8: Hover in Simulation - Mean Reward Comparison

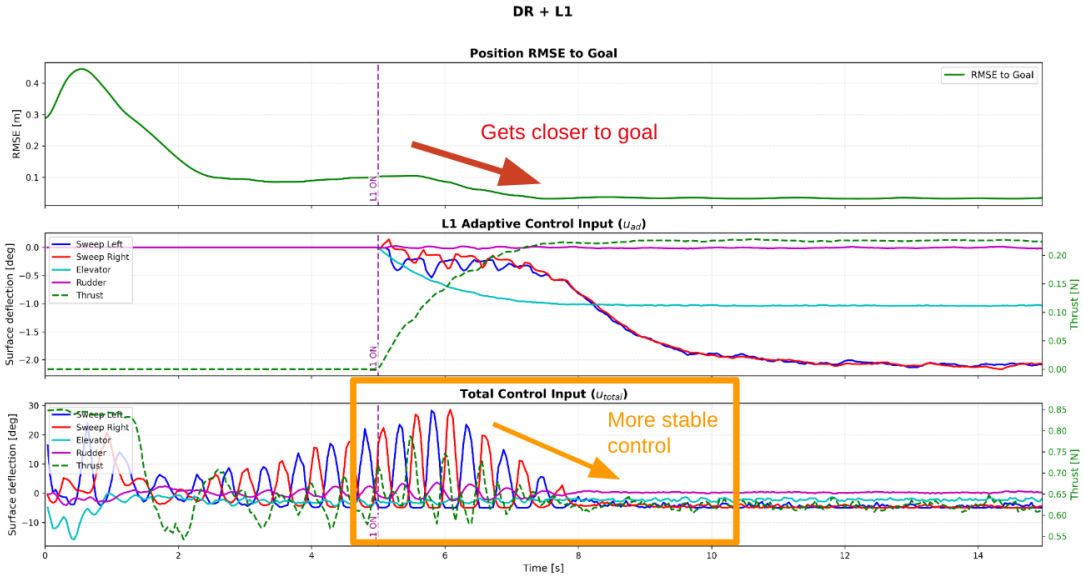


Figure 4.9: Hover in Simulation (with DR) - L1 stabilizes the control signal

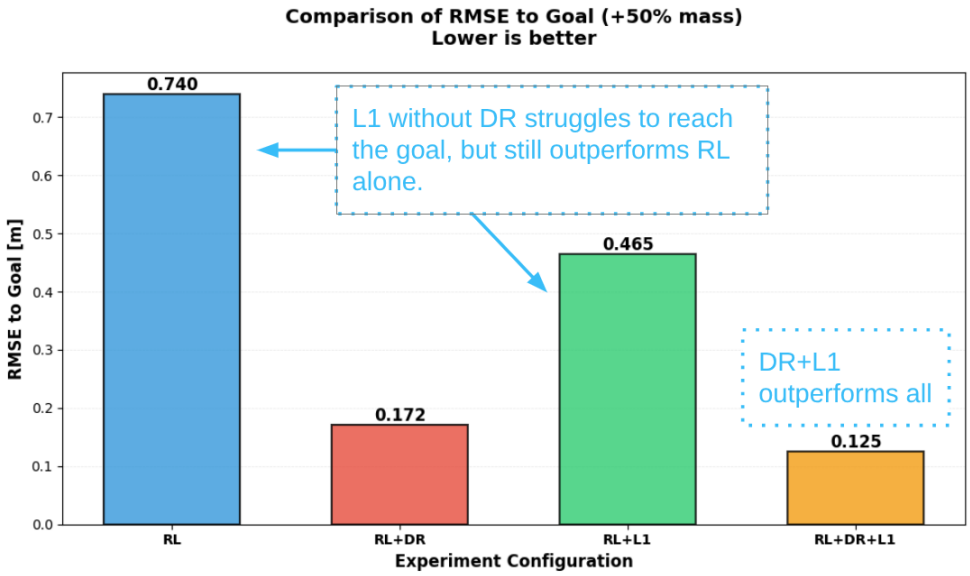


Figure 4.10: Hover in Simulation - RMSE to Goal

Figure 4.11: Hover in real-life

4.3 Agile Scenario

To further evaluate the proposed framework, experiments were conducted using an agile policy designed to execute a pitch-up maneuver. The drone is launched at an initial velocity of 6 m/s using a custom rail system. We evaluated the performance of the DR RL policy against the augmented $\mathcal{L}_1$ architecture in both simulation and physical flight tests.

4.3.1 Simulation Results

In simulation (fig. 4.13), an artificial mass offset of +20% was introduced. This offset was specifically restricted to 20% (compared to the 50% offset used in the hover scenario) because the aggressive nature of the pitch-up maneuver forces the drone to operate near its limits. With a higher mass penalty, the stringent actuator saturation constraints leave insufficient control authority for the adaptive loop to fix issues.

Despite these tight physical constraints, the $\mathcal{L}_1$-augmented policy demonstrated measurable improvements, as shown in fig. 4.12.

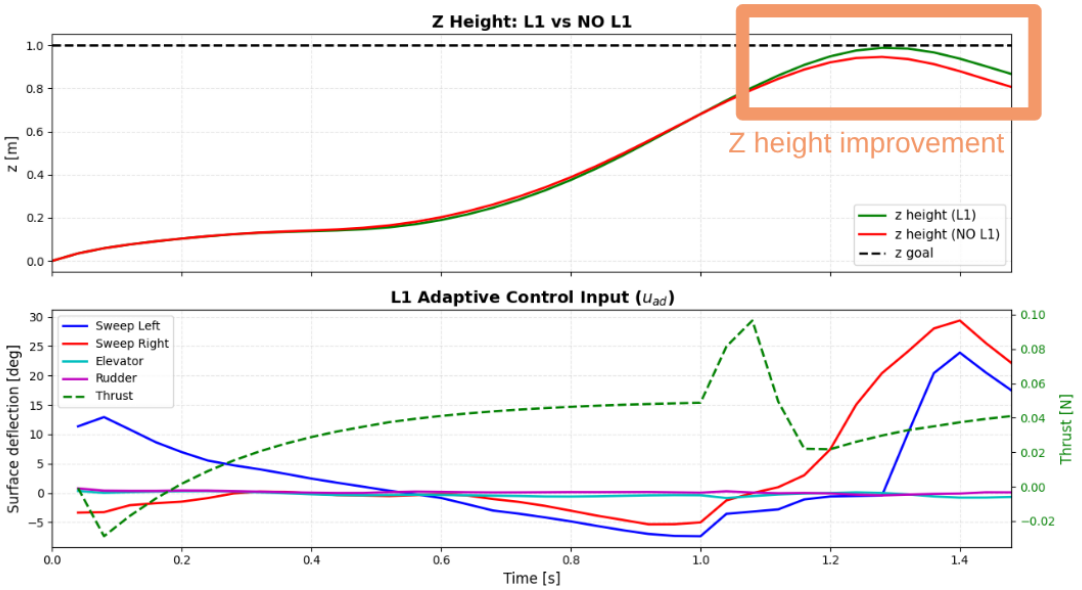


Figure 4.12: Results - Agile policy in simulation

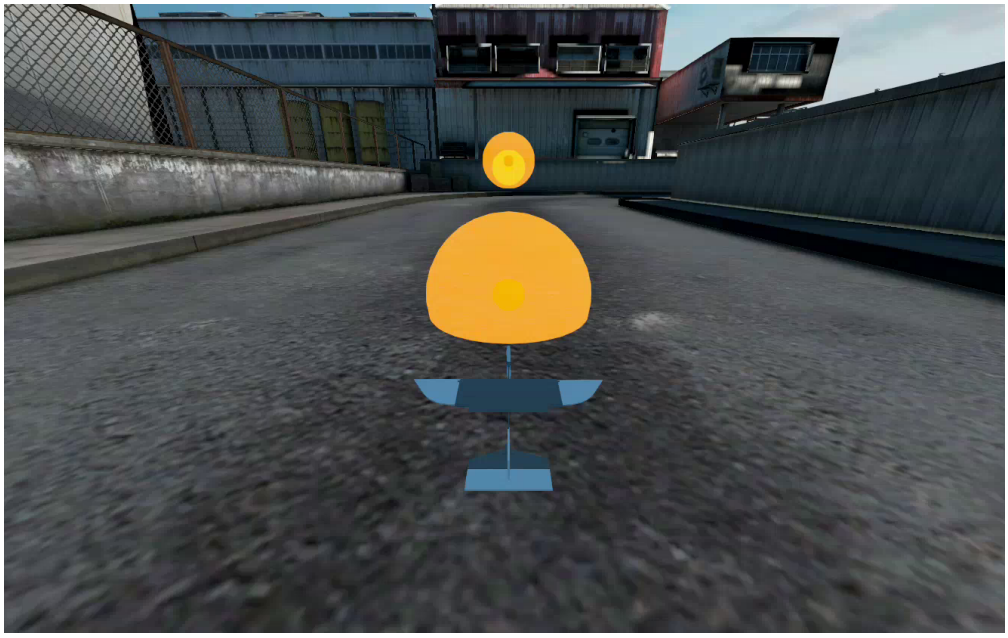


Figure 4.13: Agile policy in simulation

4.3.2 Real-World Deployment

During real-world experiments, the $\mathcal{L}_1$ controller successfully provided clear adjustments, resulting in a visible improvement in Z-height and a measurable reduction in roll error compared to the unaugmented policy (fig. 4.14). However, the real-world deployment also revealed a tendency for the adaptive control signal to overcorrect. This behavior suggests that while the adaptive augmentation improves performance, it likely requires more extensive tuning to account for the specific characteristics of the physical system.

This overcorrection is primarily attributed to hardware latency and physical actuator limits. Although the transport delay ($\tau_{act} \approx 60\,\text{ms}$) is mathematically compensated for in the state predictor (as detailed in Section 3.3.4), highly aggressive maneuvers generate prediction errors that evolve faster than the system can physically respond. The communication delay ensures the controller is always reacting to slightly aged data. This issue is amplified by the fact that the morphing servos possess inherent mechanical delays, that severely limit the execution of fast, reactive corrections. Finally, this effect is exacerbated by noise originating from the OptiTrack state estimation (particularly in the numerically differentiated velocity states).

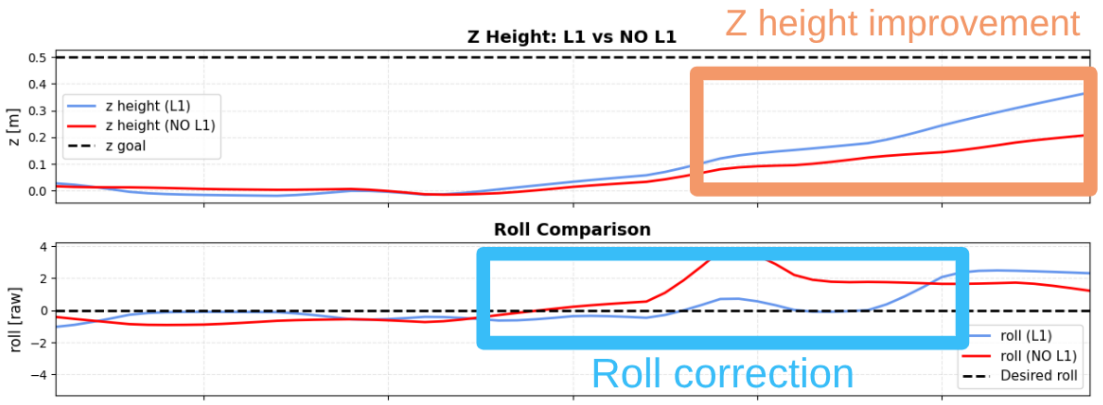


Figure 4.14: Results - Agile policy in real life

Figure 4.15: Agile flight in real-life

Chapter 5

Discussion and Limitations

5.1 Overview of the Architecture

The primary objective of this thesis was to deploy an end-to-end Reinforcement Learning policy directly to the actuators of a morphing UAV, ensuring robust real-world flight despite sim-to-real gaps. The experimental results from Chapter 4 validates that the proposed hybrid architecture successfully helps bridging this gap. By augmenting a baseline RL policy with an $\mathcal{L}_1$ Adaptive Controller, the system maintained stable flight and tracked trajectories under significant unmodeled mass offsets.

5.2 Efficacy of the End-to-End approach

A major contribution of this work is the successful circumvention of traditional cascaded control architectures. Standard approaches typically rely on RL to output high-level body rate or attitude commands to a low-level PID or INDI controller. In contrast, our framework maps observations directly to physical servo and motor commands.

The successful physical deployment of this method relies heavily on the Quadratic Programming (QP) control allocator. Morphing wings present a unique challenge

because their control authority is highly constrained and slower than a typical multicopter drone. In the hover scenarios, where we don't have very fast changes, this resulted in exceptional disturbance rejection.

5.3 Hardware bottlenecks and Limitations

Despite the success in hovering, the agile pitch-up maneuver exposed the strict physical limitations of the hardware platform. While the mathematical formulation of the $\mathcal{L}_1$ controller assumes an ideal system capable of instantaneous action, the real-world deployment was limited by multiple latency issues.

5.3.1 Latency and Actuator Bandwidth

The most prominent limitation observed was the tendency for the adaptive loop to overcorrect. This is an unavoidable consequence of operating a highly reactive 200 Hz adaptive loop over a system with a 60ms communication delay (τ_{act}).

5.4 Trade-offs in tuning

These limitations highlight a fundamental trade-off in the deployment of adaptive control on constrained mechanical systems. Achieving optimal performance requires a delicate balance between the cutoff frequency of the low-pass filter (ω), the regularization penalties ($\alpha_i, \beta_{ij}, \gamma_i$) of the QP allocator, and the values of the A_s matrix.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis presented the development, simulation, and real-world deployment of a robust, learning-based control architecture for a highly constrained morphing UAV. By combining an end-to-end Reinforcement Learning policy with an $\mathcal{L}_1$ Adaptive Control augmentation, we successfully rejected uncertainties and dynamic mismatches, proving the viability of this approach.

Crucially, this work demonstrated the feasibility of a end-to-end control paradigm. By bypassing traditional cascaded rate or attitude controllers, the RL agent was able to directly manipulate the drone's morphing geometry. While communication delays and hardware limits currently bound the maximum achievable agility, **the proposed architecture provides a robust, proven foundation for future platforms**.

6.2 Future Work

The experimental results and subsequent limitation analysis highlight several clear paths for future research and hardware development, primarily focused on alleviating the latency bottlenecks identified during physical deployment.

6.2.1 Onboard computation and sensing

The most significant limitation of the current setup is the reliance on off-board computation and state estimation. The $\approx 60\,\text{ms}$ transport delay heavily restricted the $\mathcal{L}_1$ controller's ability to safely reject fast disturbances. Future iterations of this platform must transition the control architecture to an onboard companion computer. Additionally, replacing OptiTrack motion capture system with an onboard pipeline might eliminate extra noise and latency.

6.2.2 Wing-Twist mechanism for better actuation

As observed during the pitch-up maneuver, the highly limited actuation of the current platform frequently leads to severe actuator saturation. When the adaptive loop commands aggressive corrections, the existing control surfaces simply lack the required control authority and hit their physical limits, which degrades flight performance.

To directly mitigate this, future work will focus on deploying the proposed control strategies on a new (work-in-progress) prototype equipped with a wing-twist mechanism (see fig. 6.1). Introducing wing twist will significantly expand the platform's aerodynamic control authority.

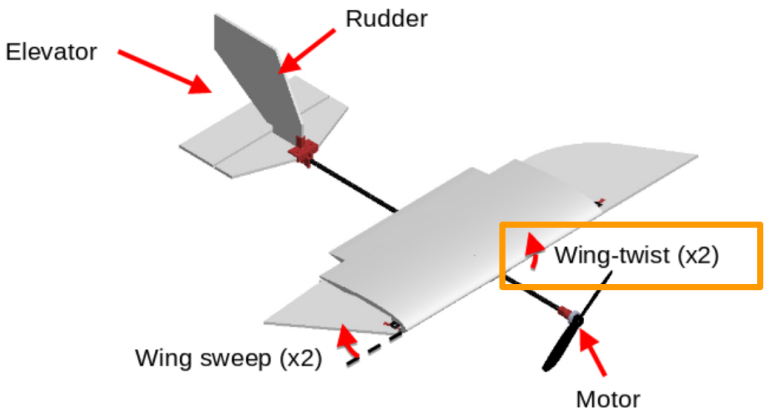


Figure 6.1: Future drone with wing twist

6.2.3 Integration of INDI

While the current $\mathcal{L}_1$ adaptive architecture successfully manages parametric uncertainties and unmodeled mass penalties, its performance during fast maneuvers can be further augmented by integrating Incremental Nonlinear Dynamic Inversion (INDI) [18]. Recent literature demonstrates that combining INDI with $\mathcal{L}_1$ yields superior disturbance rejection and increases systemic robustness in unpredictable environments [19] (see fig. 6.2).

These two control strategies naturally complement each other. INDI uses direct angular acceleration feedback to instantly cancel fast, transient disturbances like wind gusts. Meanwhile, the $\mathcal{L}_1$ controller is specifically designed to compensate for slower and more persistent model uncertainties, such as mass offsets. Combining them would theoretically provide the morphing drone with comprehensive robustness across a much wider range of flight conditions.

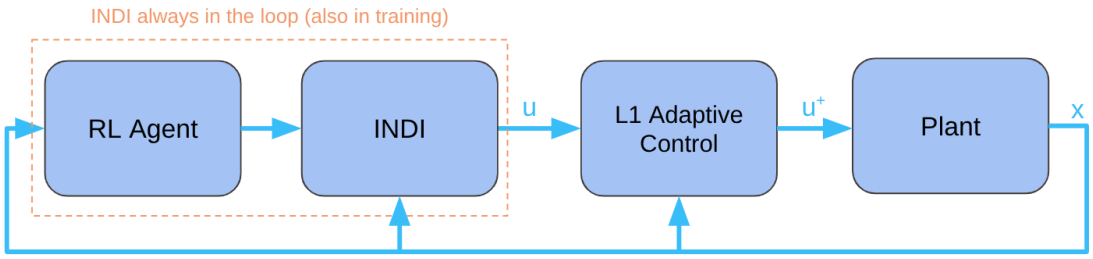


Figure 6.2: Combined RL+INDI+L1 architecture

Appendix A

Code structure

A.1 Code structure of the RL+$\mathcal{L}_1$ architecture

In the following section we'll highlight some of the core code that was developed for the thesis. The code snippets will be simplified for explanation clarity.

A.1.1 Overview

As described already we have an hierarchical and multi-rate architecture:

- **High-level policy (RL)** computes a nominal command u_{HL} at period dt_{HL}.
- **Low-level L1 controller** runs faster at dt_{LL}, estimates disturbance/mismatch, and applies adaptive correction u_{ad}.

The resulting command is:

$$u_{\text{total}} = \text{clamp}(u_{HL} + u_{ad})\,. \tag{A.1}$$

The code is made of a mix of **C++** and **Python**, bridged by **pybind11**, which enables interfacing between the two languages. The entrypoint and all the RL stack is written in Python, while all the rest (environment, simulation, dynamics,

controllers) is written in C++.

Some of the hyperparameters and configuration variables are stored in YAML files that the user can customize. An example snippet can be seen in Listing 1. Next sections highlights some of the core code snippets.

```yaml
environment:
  world_box: [-100, -100, -100, 100, 100, 200]

simulation:
  ctr_dt: 0.04
  ctr_dt_ll: 0.005
  max_t: 7

robot:
  ll_controller: L1

L1:
  ctr_dt_B: 0.1
  omega_lpf: 1.0
  As_diagonal: [-5.0, -5.0, -5.0, -10.0, -10.0, -10.0]
  qp_asym_pairs_penalty: [[1,2,0.1]]
  qp_rate_penalty: 0.1
  qp_energy_penalty: 0.01
  activation_time: 0.0
```

Listing 1: Typical L1 configuration

A.1.2 L1 Core Update

```
u_HL = cmd.getRaw();                          // Get RL command

// ...Run delay compensation...

x_tilde = x_hat - x_vw_correct;               // estimation error
sigma_hat = sigma_gain * x_tilde;             // adaptation law

// Update B with a reduced frequency (10Hz)
if (t == 0.0 || std::fmod(t + 1e-9, dt_B) < dt) {
    B_control = get_B_control(state_correct);
}

// Use QP allocation to get sigma_hat_m
sigma_hat_m = get_sigma_hat_m_QP(B_control, sigma_hat, u_HL);

// Low-pass filtered adaptive command
u_ad = -((omega_lpf*dt)*sigma_hat_m + (1 - omega_lpf*dt)*(-u_ad));

// Final command with saturation
u_total = dynamics_ptr_->clampRaw(u_ad + u_HL);

// predictor update
Vector<X_VW_DIM> vw_dot = dynamics_ptr_->getAccelerations(
    state_correct, u_total, dt, wind_curl,
    cfg_["L1"]["state_predictor_skip_actuator_dynamics"].as<bool>(false)
).head<6>();

x_hat = x_hat + (vw_dot + sigma_hat + As * x_tilde)*dt;
```

Listing 2: Core of the L1 adaptive update

A.1.3 QP Allocation

```
// Objective: min ||Bx - y||^2  =>  min 1/2 x'Px + q'x
H_row.noalias() = 2.0 * B.transpose() * B;
g_buffer.noalias() = -2.0 * B.transpose() * sigma_hat;

 // Regularization
H_row.diagonal().array() += 1e-9;

// If configured, add alpha_i * ||x_i - x_prev_i||^2 to the objective.
if (qp_rate_penalty.size() == x_act_dim) {
    H_row.diagonal().array() += 2.0 * qp_rate_penalty.array();
    g_buffer.noalias() += (-2.0 * (qp_rate_penalty.array()
                        * sigma_hat_m_QP_prev.array())).matrix();
}

// ...

int nWSR = 50;
qpOASES::returnValue ret;
ret = solver_ptr->init(
    H_row.data(),          // Hessian (must be RowMajor!)
    g_buffer.data(),       // Gradient
    NULL,                  // Constraint Matrix A (none), we only have bounds
    u_min_buffer.data(),   // Lower Bounds Array
    u_max_buffer.data(),   // Upper Bounds Array
    NULL,                  // Lower Bounds for A (none)
    NULL,                  // Upper Bounds for A (none)
    nWSR                   // nWSR = Maximum number of working set recalculations
);
```

Listing 3: Simplified QP setup for control allocation

A.2 ROS interface for real-life tests

As mention in Chapter 4, we use a ROS setup to control the drone in real-life. The workspace has 2 nodes: one to handle the control loops and another to receive and

convert motion-capture poses from VRPN to a ROS-compatible format.

A.2.1 Main ROS Node

```
def main():
    filtered_args = [arg for arg in sys.argv if not arg.startswith('__')]
    args = parser().parse_args(filtered_args[1:])

    cfg = get_config(...)
    make_deterministic(cfg)

    # load latest PPO trial/iter if not provided
    # load policy + RMS
    yaml_string = clean_and_dump_yaml(cfg)
    eval_env = RobotEnv_v1(yaml_string, False, False)
    eval_env = wrapper.FlightEnvVec(eval_env)
    eval_env.load_rms(rms_path)

    saved_variables = torch.load(weight_path, map_location=device)
    policy = MlpPolicy(**saved_variables["data"])
    policy.action_net =
            torch.nn.Sequential(policy.action_net, torch.nn.Tanh())
    policy.load_state_dict(saved_variables["state_dict"], strict=False)
    policy.to(device)

    rospy.init_node("flightpilot", anonymous=True)
    _ = RosInterface(rl_policy=policy, env=eval_env, cfg=cfg, args=args)
    rospy.spin()
```

Listing 4: ROS pilot entry: load policy and start interface

A.2.2 ROS Control Loops (inside *RosInterface*)

```
self.HL_DT = 0.04
self.LL_DT = 0.005

self.hl_control_loop_timer = rospy.Timer(
    rospy.Duration(0, int(self.HL_DT * 1e9)), self.hl_control_loop
)
self.ll_control_loop_timer = rospy.Timer(
    rospy.Duration(0, int(self.LL_DT * 1e9)), self.ll_control_loop
)

# ...

def hl_control_loop(self, timer):
    robot_state = self.pose_estimator.get_state()
    if robot_state is None:
        return
    obs, self.last_reward, self.is_done, info =
        self.env.setRobotState(robot_state)
    self.cmd_hl =
        self.rl_policy.predict(
            np.asarray(obs),
            deterministic=True
        )[0].flatten()
    self.env.step(self.cmd_hl)

def ll_control_loop(self, timer):
    robot_state = self.pose_estimator.get_state()
    if robot_state is None or self.is_done or self.cmd_hl is None:
        return

    self.env.setLLReference(self.cmd_hl, wind)
    self.cmd_ll = self.env.callLLRun(robot_state)[0][0:self.num_actions]
    self.cmd_ll = self.renormalize_and_clamp(self.cmd_ll)

    # Turn off motor and reset actuators if out of world box
    if self.is_in_world_box(robot_state):
        self.arduino_interface.send(self.cmd_ll)
    else:
        self.arduino_interface.send(self.final_action)
```

Listing 5: ROS high-level and low-level loops

A.2.3 Arduino Interface

The hardware bridge sends low-level commands from ROS to the onboard Arduino via UDP. The command vector is first transformed and serialized.

```
import socket
import numpy as np

class ArduinoInterface:
    def __init__(self):
        self.UDP_IP = "192.168.194.234"
        self.UDP_PORT = 2390
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    def send(self, con):
        con[1] = -con[1]  # left sweep sign convention
        str_con = self._encode(con)
        self._call(str_con)

    def _encode(self, con):
        con = np.array(con)
        res = 3
        con = np.round(self._compress(con), res)

        str_con = str()
        for i in range(len(con)):
            con_clipped = np.clip(con[i], 10**-res, 1-10**-res)
            str_con += str(con_clipped)[1:].replace('.', '').ljust(res, "0")
        return str_con

    def _compress(self, con):
        return con / 2 + 0.5

    def _call(self, con):
        self.sock.sendto(
            bytes(str(con), "utf-8"),
            (self.UDP_IP, self.UDP_PORT)
        )
```

Listing 6: ROS-to-Arduino UDP command interface

A.2.4 Motion Capture

This ROS node converts mocap poses from VRPN format to the pose format used by ROS.

```
def pose_callback(self, pose_msg, rigid_body_name):
    pose = PoseStamped()
    pose.header.frame_id = 'world'
    pose.header.stamp = rospy.Time.now()

    # position
    pose.pose.position.x = pose_msg.pose.position.z
    pose.pose.position.y = pose_msg.pose.position.x
    pose.pose.position.z = pose_msg.pose.position.y

    # orientation
    pose.pose.orientation.x = pose_msg.pose.orientation.z
    pose.pose.orientation.y = pose_msg.pose.orientation.x
    pose.pose.orientation.z = pose_msg.pose.orientation.y
    pose.pose.orientation.w = pose_msg.pose.orientation.w

    self.publishers[rigid_body_name].publish(pose)
```

Listing 7: Core frame transformation callback

A.2.5 Pose Estimator

The pose estimator builds the full robot state from mocap measurements:

- Subscribes to the converted MoCap topics.
- Converts pose to internal coordinates.
- Computes filtered body velocity and angular velocity.
- Computes filtered linear and angular accelerations by differentiation.
- Returns state vector:

$$x = [p,\ q,\ v,\ \omega,\ a,\ \alpha].$$

```
# ...

def get_state(self):
    if not self.is_mocap_pose_initialized:
        return None
    return np.concatenate(
        [self.pos, self.quat, self.vel, self.omega, self.acc, self.aac],
        axis=0
    ).astype(np.float64)

def _update_derived_states(self, dt):
    self.att_rate_euler =
        (self.att_euler - self.att_euler_prev) / dt
    self.omega_new =
        euler_to_body_rate(self.att_euler, self.att_rate_euler)
    self.vel_new =
        np.matmul(self.R_global_to_body, (self.pos - self.pos_prev) / dt)

    self.vel =
        lpf(self.vel_new, self.vel.copy(), self.f_cut_vel, dt)
    self.omega =
        lpf(self.omega_new, self.omega.copy(), self.f_cut_omega, dt)

    if not self.acc_inits:
        self.vel_prev = self.vel
        self.omega_prev = self.omega
        self.acc, self.aac = np.zeros(3), np.zeros(3)
        self.acc_inits = True
    else:
        self.acc_new_unfilt = (self.vel - self.vel_prev) / dt
        self.aac_new_unfilt = (self.omega - self.omega_prev) / dt
        self.acc =
            lpf(self.acc_new_unfilt, self.acc, self.f_cut_acc, dt)
        self.aac =
            lpf(self.acc_new_unfilt, self.aac, self.f_cut_aac, dt)
        self.vel_prev = self.vel
        self.omega_prev = self.omega
```

Listing 8: Pose Estimation from MoCap data

A.3 Additional tools

Additional tools and scripts have been created to facilitate the analysis and debugging of the system. For example, after every run flightmare generates a CSV file with all the data (controllers, policy, state estimation, ...). A python program was made to easily visualize this data (`plot_data_L1_pyqt.py`). Screenshots of this software are shown below.

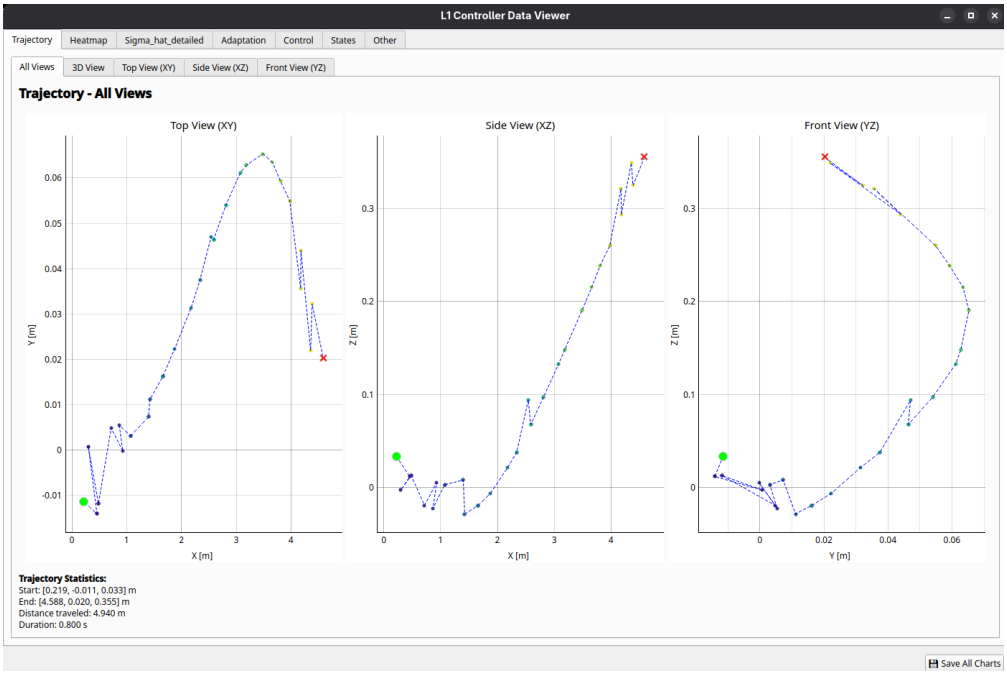


Figure A.1: Data Viewer

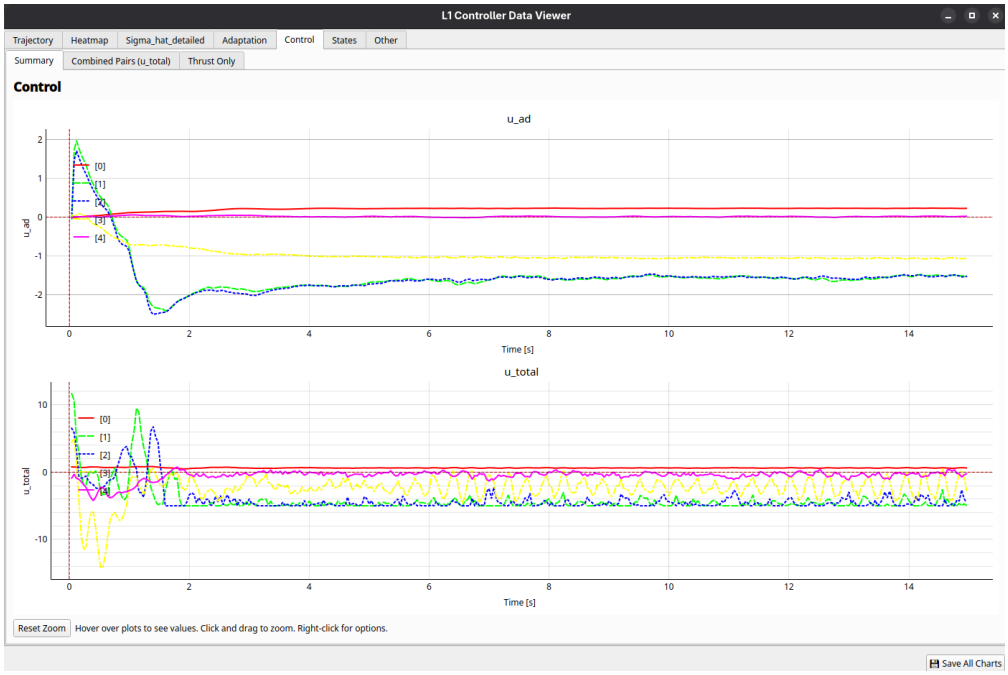


Figure A.2: Data Viewer

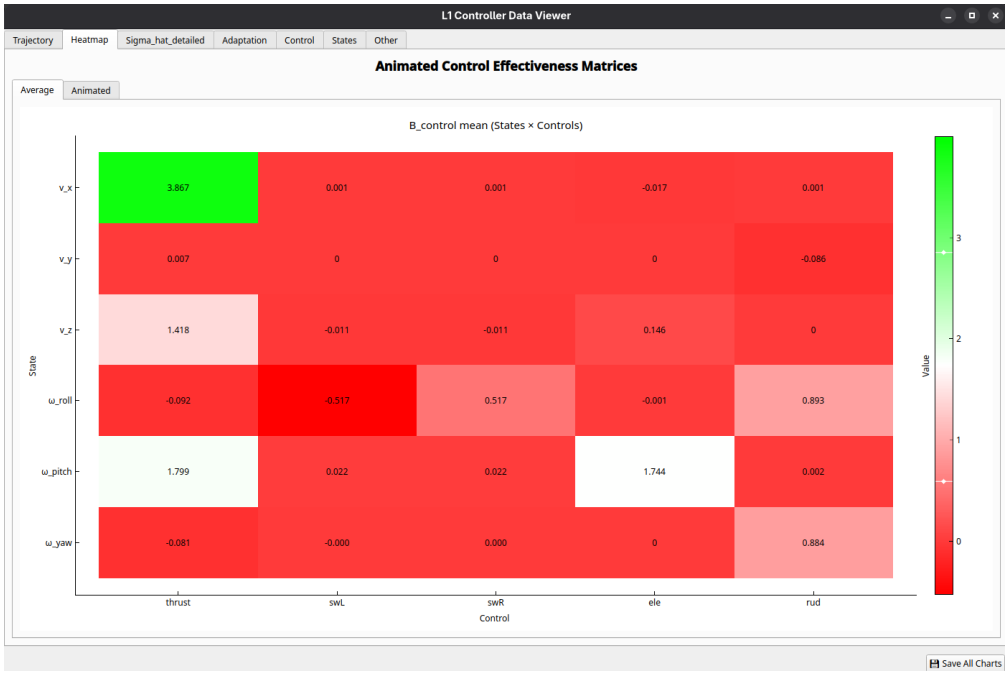


Figure A.3: Data Viewer

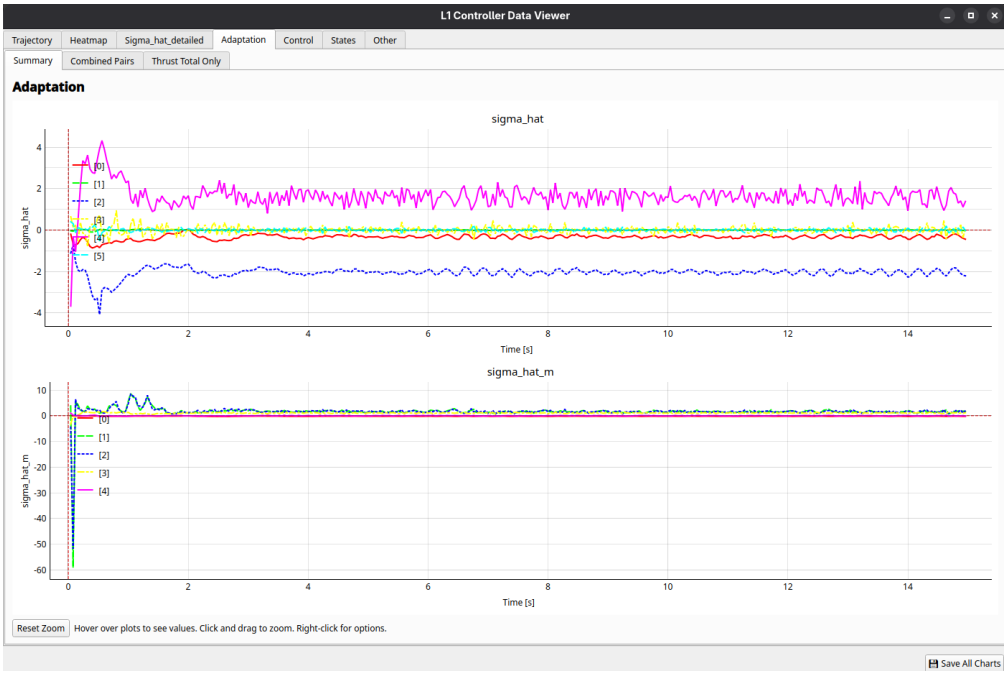


Figure A.4: Data Viewer

Bibliography

[1] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 79767976, pp. 982–987, Aug. 2023. DOI: 10.1038/s41586-023-06419-4.

[2] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1205–1212. DOI: 10.1109/IROS51168.2021.9636053.

[3] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *CoRR*, vol. abs/1703.06907, 2017. arXiv: 1703.06907. [Online]. Available: http://arxiv.org/abs/1703.06907.

[4] K. J. Astrom and B. Wittenmark, *Adaptive Control*, 2nd. USA: Addison-Wesley Longman Publishing Co., Inc., 1994, ISBN: 0201558661.

[5] N. Hovakimyan and C. Cao, *L1 Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation* (Advances in Design and Control). Society for Industrial and Applied Mathematics, 2010, ISBN: 9780898717044.

[6] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 690–697, 2022. DOI: 10.1109/LRA.2021.3131690.

[7] Y. Liu, F. Quan, and H. Chen, *Adaptive nonlinear mpc for trajectory tracking of an overactuated tiltrotor hexacopter*, 2022. arXiv: 2211.06762 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2211.06762.

[8] Z. Wu et al., "$\mathcal{L}_1$ Adaptive Augmentation for Geometric Tracking Control of Quadrotors," en, in *2022 International Conference on Robotics and Automation (ICRA)*, arXiv:2109.06998 [eess], May 2022, pp. 1329–1336. DOI: 10.1109/ICRA46639.2022.9811946. Accessed: Aug. 6, 2025. [Online]. Available: http://arxiv.org/abs/2109.06998.

[9] S. Snyder, P. Zhao, and N. Hovakimyan, "L1 Adaptive Control with Switched Reference Models: Application to Learn-to-Fly," en, *Journal of Guidance, Control, and Dynamics*, vol. 45, no. 12, pp. 2229–2242, Dec. 2022, ISSN: 0731-5090, 1533-3884. DOI: 10.2514/1.G006305. Accessed: Aug. 6, 2025. [Online]. Available: https://arc.aiaa.org/doi/10.2514/1.G006305.

[10] L. Cheng, Y. Li, J. Yuan, J. Ai, and Y. Dong, "L1 Adaptive Control Based on Dynamic Inversion for Morphing Aircraft," en, *Aerospace*, vol. 10, no. 9, p. 786, Sep. 2023, ISSN: 2226-4310. DOI: 10.3390/aerospace10090786. Accessed: Jan. 19, 2024. [Online]. Available: https://www.mdpi.com/2226-4310/10/9/786.

[11] B. Michini and J. How, "L1 Adaptive Control for Indoor Autonomous Vehicles: Design Process and Flight Testing," en, in *AIAA Guidance, Navigation, and Control Conference*, Chicago, Illinois: American Institute of Aeronautics and Astronautics, Aug. 2009, ISBN: 978-1-60086-978-5. DOI: 10.2514/6.2009-5754. Accessed: Aug. 6, 2025. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2009-5754.

[12] Z. Wu et al., *$\mathcal{L}_1$Quad: $\mathcal{L}_1$ adaptive augmentation of geometric control for agile quadrotors with performance guarantees*, 2024. arXiv: 2302.07208 [eess.SY]. [Online]. Available: https://arxiv.org/abs/2302.07208.

[13] S. L. Jeger, V. Wüest, C. Toumieh, and D. Floreano, "Adaptive morphing of wing and tail for stable, resilient, and energy-efficient flight of avian-inspired drones," *npj Robotics*, vol. 2, no. 1, Nov. 2024, ISSN: 2731-4278. DOI: 10.1038/s44182-024-00015-y. [Online]. Available: http://dx.doi.org/10.1038/s44182-024-00015-y.

[14] V. Wüest, S. Jeger, M. Feroskhan, E. Ajanic, F. Bergonti, and D. Floreano, "Agile perching maneuvers in birds and morphing-wing drones," *Nature Communications*, vol. 15, no. 1, p. 8330, 2024, ISSN: 2041-1723. DOI: 10.1038/s41467-024-52369-4. [Online]. Available: https://doi.org/10.1038/s41467-024-52369-4.

[15] Y. Cheng, P. Zhao, F. Wang, D. J. Block, and N. Hovakimyan, "Improving the robustness of reinforcement learning policies with $\mathcal{L}_1$ adaptive control," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6574–6581, 2022. DOI: 10.1109/LRA.2022.3169309.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347.

[17] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," *CoRR*, vol. abs/2103.08624, 2021. arXiv: 2103.08624. [Online]. Available: https://arxiv.org/abs/2103.08624.

[18] A. STEINERT, S. RAAB, S. HAFNER, F. HOLZAPFEL, and H. HONG, "From fundamentals to applications of incremental nonlinear dynamic inversion: A survey on indi – part i," *Chinese Journal of Aeronautics*, vol. 38, no. 11, p. 103 553, 2025, ISSN: 1000-9361. DOI: https://doi.org/10.1016/j.cja.2025.103553. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1000936125001591.

[19] G. Wang, Z. Wei, Z. Zuo, H. Zhan, and P. Li, "Adaptive augmentation of incremental nonlinear dynamic inversion for micro quadrotor aerobatic flight control," *IEEE Transactions on Aerospace and Electronic Systems*, pp. 1–12, 2026. DOI: 10.1109/TAES.2026.3663140.

Use of AI Tools

Large Language Models (LLMs) were used during the preparation of this thesis to assist with language refinement, restructuring of text, and clarification of explanations. All technical content, interpretations, results, and conclusions were produced by the author, who takes full responsibility for their correctness.

Acknowledgements

I would like to express my sincere gratitude to Prof. Dario Floreano for welcoming me into the lab and providing me with the opportunity to conduct this research.

A special thanks goes to my supervisors, Julius Wanner and Simon Jeger, for your invaluable guidance, patience, and continuous support throughout this project. Thank you for the insightful discussions, for helping me navigate the challenges, and for the constructive feedback that greatly improved this thesis.

I would also like to thank the entire LIS team for creating such a stimulating and friendly working environment.

Lausanne, Jan 2026