Data-Efficient Deep Reinforcement Learning for Attitude Control of Fixed-Wing UAVs: Field Experiments

Eivind Bøhn, Erlend M. Coates, Dirk Reinhardt, and Tor Arne Johansen

Attitude control of fixed-wing unmanned aerial vehicles (UAVs) is a difficult control problem in part due to uncertain nonlinear dynamics, actuator constraints, and coupled longitudinal and lateral motions. Current state-of-the-art autopilots are based on linear control and are thus limited in their effectiveness and performance. Deep reinforcement learning (DRL) is a machine learning method to automatically discover optimal control laws through interaction with the controlled system, which can handle complex nonlinear dynamics. We show in this paper that DRL can successfully learn to perform attitude control of a fixed-wing UAV operating directly on the original nonlinear dynamics, requiring as little as three minutes of flight data. We initially train our model in a simulation environment and then deploy the learned controller on the UAV in flight tests, demonstrating comparable performance to the state-of-the-art ArduPlane proportional-integral-derivative (PID) attitude controller with no further online learning required. Learning with significant actuation delay and diversified simulated dynamics were found to be crucial for successful transfer to control of the real UAV. In addition to a qualitative comparison with the ArduPlane autopilot, we present a quantitative assessment based on linear analysis to better understand the learning controller's behavior.

Index Terms—**Deep reinforcement learning, Autonomous aerial vehicles, Attitude control, Sim-to-real, Soft actor critic**

I. INTRODUCTION

Many challenging control problems arise during advanced operation of fixed-wing unmanned aerial vehicles (UAVs), such as aerobatic maneuvering [1], perching [2], deep-stall landing [3], recovery from loss of control [4], flying in strong wind fields [5], or performing VTOL transitions between hover and forward flight [6]. Fixed-wing UAVs, as illustrated in Fig. 1, have superior range and endurance when compared to multirotor UAVs. However, the control of such vehicles is challenging due to highly coupled, underactuated nonlinear dynamics, as well as uncertain aerodynamics affected by wind disturbances that make up a large fraction of the vehicle's airspeed.

A class of methods that have shown promising results for challenging control problems is deep reinforcement learning (DRL). Reinforcement learning (RL) is an area of machine learning concerned with learning optimal sequential decision-making. DRL is the combination of RL algorithms with

neural networks (NNs) as function approximators, which is the state-of-the-art approach for many problems requiring complex decision making over long time horizons such as game-playing [7], dexterous in-hand robotic manipulation [8], and object manipulation [9]. It can handle continuous state and action spaces, highly complex and nonlinear system dynamics, and in general, does not require a model of the system to be controlled. Furthermore, the online execution of an RL controller is often very computationally efficient. This should make DRL an enticing alternative for problems where accurate identification of the system is difficult and traditional control approaches are unable to yield sufficient control performance. Despite this potential, DRL has yet to be widely adopted for control and notably lacks demonstrations of control applications outside of simulations. One of the main contributing factors to this is the lack of safety guarantees and the ability to formulate operating constraints, both in the exploration and exploitation phase, which is further complicated by the data-intensive nature of DRL. We will in the rest of this paper use RL to refer to DRL.

An approach to mitigate the challenges of RL for control is foregoing online exploration entirely and learning the controller exclusively from historical data with no further interaction with the system to be optimized, i.e. offline RL [10]. The latter is a radical alteration of the RL problem introducing new challenges and necessitating its own set of learning algorithms. It could nevertheless be an important tool in the future for problems such as control of UAVs — where data collection carries a high risk and accurate modeling is difficult. A more common approach is performing part of or the whole exploration phase in a simulation of the target system. A downside of this approach is that while RL in principle is a model-free optimization framework, the success of the transfer from the simulated environment to the real target environment is highly affected by the accuracy of the simulation model, the lack of which is referred to as the reality gap in RL. One should therefore take great effort in minimizing the reality gap through sim-to-real measures, which aim at robustifying the learned controller and emulating effects such as latency and measurement noise present in the real control system. For a recent survey on sim-to-real methods in the context of control and robotics, see [11].

For the sim-to-real learning approach to be useful for practical flight control applications, controllers trained in simulation need to transfer well to control the real UAV. Before attempting advanced problems like e.g. deep-stall landing, it makes sense to first attempt simpler problems and identify what factors are important for controllers to transfer well to

Eivind Bøhn is with the Department of Mathematics and Cybernetics, SINTEF DIGITAL, Oslo, Norway (e-mail: eivind.bohn@sintef.no)

Erlend M. Coates, Dirk Reinhardt, and Tor Arne Johansen are with the Centre for Autonomous Marine Operations and Systems, Department of Engineering Cybernetics, NTNU, Trondheim, Norway (e-mail: {erlend.coates, dirk.reinhardt, tor.arne.johansen}@ntnu.no)

Fig. 1. The Skywalker X8 fixed-wing aircraft in flight.

reality. In this paper, we consider the attitude control problem of fixed-wing UAVs. Attitude refers to the orientation of the aircraft, and control of the attitude constitutes the lowest level of flight control deciding how the actuators of the UAV are used to achieve the desired attitude as decided by the guidance components of the control system. This work is a follow-up to our previous work [12] in which we demonstrate the efficacy of DRL for attitude control of fixed-wing UAVs in a simulator environment. We now target control of the real UAV in the field and develop a framework to learn attitude controllers with a focus on data efficiency, yielding flightworthy controllers with only minutes of learning time. Starting with a UAV model obtained primarily through wind-tunnel experiments, we adopt the method of exploring and learning in a simulator environment and iteratively adjust the model and simulation environment with insights from flight experiments. We extensively apply domain randomization and other sim-to-real measures in order to reduce the reality gap. Moreover, the data efficiency of our method limits overfitting to the simulation model, such that the controller transfers better to the field, and when combined with safe exploration measures the high data efficiency could enable learning controllers entirely on the real UAV in the field.

The literature on RL-based attitude control of UAVs is dominated by quadcopters, and most works operate exclusively in simulated environments [13, 14, 15, 16, 17]. When it comes to works presenting real-world flight experiments we have identified the following: [18, 19, 20, 21, 22, 23, 24]. Of these, only [21], their follow-up work [22], and [23] use a fixed-wing UAV design. [21] and [22] study the specific problem of controlling a perched landing, [23] fixes the aircraft in a wind-tunnel and limit themselves to controlling the pitch of the UAV, while we study the attitude control problem of an unconstrained vehicle in a 3D outdoor environment. Moreover, the data requirement of their methods is on the order of millions of samples. The other aforementioned works also require millions of samples of data, with the notable exception of [18]. Their method uses model-based RL and involves learning a forward dynamics model that is used in a Model Predictive Control (MPC) scheme which controls the quadrotor. While this method is very sample efficient, the MPC is too computationally complex to run aboard the UAV and therefore necessitates continuous communication with an external computer, while our controller can run on a fraction of the computational power embedded in the UAV. For a more general overview of the application of RL to UAVs see [25]

and the related works section of [12].

Other related control methods that have been proposed for attitude control of fixed-wing UAVs include simple proportional-integral-derivative (PID) loops [26], the linear quadratic regulator (LQR) [27], adaptive dynamic programming (ADP) [28, 29, 30], and MPC [31]. ADP is similarly to RL a data-driven optimal control scheme. In the works of [28, 29, 30], the cost function is assumed to have a quadratic form and the optimal control law is derived from the Hamilton–Jacobi–Bellman (HJB) equation, which is solved numerically by approximating the cost-function using a value iteration scheme. Of note is also MPC which has inherent support for system constraints, multivariate objectives, high interpretability, and the ability to incorporate domain knowledge in the dynamics model. Motivated by these properties, the authors of this work have developed an MPC controller for simultaneous control of the UAV's attitude and airspeed, allowing the controller to consider the coupling between pitch and airspeed dynamics (note that we have previously demonstrated that RL is also capable of learning this coupled control problem [12, 32]). Flight experiments with this MPC were conducted using the same experimental platform as this work, albeit with a more powerful processing unit. See [31] for details and experimental results, as well as the discussion in [33]. Due to the differences in control objective and hardware platform, an in-depth comparison between MPC and the RL controller is outside the scope of this paper.

The contributions and novelty of this paper can be summarized as follows:

- To the best of our knowledge, this is the first work to demonstrate through field experiments the efficacy of RL for attitude control of fixed-wing UAVs, a class of UAV design generally considered to be significantly more complex to control than the multirotor which is common in the literature.
- The proposed method improves upon the data efficiency of the existing literature by at least an order of magnitude. We show that our method can develop flightworthy controllers with only 3 minutes of data from interaction with the simulation environment, providing an important step towards enabling the learning of RL controllers entirely on the real UAV.
- We present an analysis of the RL controller in order to better understand how it operates, including a comparison to an industry-standard PID controller.

The rest of this paper is organized as follows: Section II presents the RL framework used to optimize the controller. Section III describes our method in detail and presents our thoughts on what parameters are important for the learning problem. Section IV presents the main experimental results obtained in real flight, followed by a discussion and analysis part in Section V. Finally, Section VI concludes with our thoughts on the presented work and ideas for future research.

II. REINFORCEMENT LEARNING

The RL optimization framework consists of two main parts, an environment that is to be controlled, and an agent that

observes the state of the environment and selects actions to maximize the rewards it receives. The environment is typically described as a Markov decision process (MDP), which is defined by a 5-tuple of components $(\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$: A set of states, $\mathcal{S}$, a set of actions available to the agent, $\mathcal{A}$, a transition function $\mathcal{T}(s_t, a_t) = s_{t+1}$ which describe the evolution of the states as a function of actions and previous states, a reward function $R(s, a)$ quantifying the utility of states and accompanying actions, and finally, a discount factor $\gamma \in [0, 1)$, weighing the relative value of immediate and future rewards.

We consider in this paper the episodic finite-horizon formulation of RL. An episode is a trajectory of states and actions $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$ of maximum length T with a distribution of initial states $\mathcal{S}_0$. The policy π_ϑ is a parameterized function that maps states to actions, describing the agent's behaviour (analogous to a control law in control terminology). The RL objective can then be stated as finding the optimal parameters ϑ of the policy π_ϑ^* that maximizes the return G over the distribution of the initial conditions of the episode:

$$G(\tau) = \sum_{t=0}^{T} \gamma^t R(s_t, a_t) \tag{1}$$

$$\pi_\vartheta^* = \underset{\vartheta}{\arg\max} \, \mathbb{E}_{\tau \sim \mathcal{T}(\mathcal{S}_0, \pi_\vartheta)} [G(\tau)] \tag{2}$$

where $\sim$ signifies that the trajectories τ are sampled from $\mathcal{T}(\mathcal{S}_0, \pi_\vartheta)$, i.e. the distribution of trajectories given by the transition dynamics $\mathcal{T}$, the initial state distribution $\mathcal{S}_0$, and the action-distribution of the policy π_ϑ.

Soft Actor Critic (SAC) [34] is an actor-critic algorithm whose defining characteristic is its entropy-regularization, meaning that it is jointly maximizing the expected rewards as in (2) and maximizing the entropy of the policy:

$$\pi_\vartheta^* = \underset{\vartheta}{\arg\max} \, \mathbb{E}_{\tau \sim \mathcal{T}(\mathcal{S}_0, \pi_\vartheta)} \left[\sum_{t=0}^{T} \gamma^t \left(R(s_t, a_t) + \chi \mathcal{H}(\vartheta | s_t, a_t) \right) \right] \tag{3}$$

$$\vartheta_{\text{new}} = \vartheta_{\text{old}} + \eta \nabla_\vartheta J^{\text{SAC}}(\pi_\vartheta) \tag{4}$$

where $\mathcal{H}(\vartheta | s, a) = \mathbb{E}_{a \sim \pi_\vartheta(a|s)} [-\log \pi_\vartheta(a|s)]$ is the entropy, equal to the negative log probability of the action-distribution of the policy in the state in question, χ is the entropy coefficient, and (4) shows the gradient ascent scheme used to arrive at the optimal policy in which $\eta > 0$ is the learning rate and J^{SAC} is the SAC objective function. For brevity, we limit our discussion about SAC to the implementation of the policy and instead refer the reader to the original paper [34] for details on the objective function. The policy is implemented as follows:

$$\pi_\vartheta(s_t) = \tanh(\mu_\vartheta(s) + \sigma_\vartheta(s) \odot \xi), \; \xi \sim \mathcal{N}(0, I) \tag{5}$$

here, μ_ϑ and σ_ϑ are two parameterized deterministic functions of the input, representing the mean and covariance of the output, respectively. The notation $\odot$ denotes element-wise matrix-multiplication and ξ is independently sampled Gaussian noise. The entropy can therefore be controlled in a state-dependent manner through the σ_ϑ function. Finally, the output is saturated with the hyperbolic tangent function, which squashes the Gaussian's infinite support to the domain $[-1, 1]$, limiting the adverse effects of extreme noise values and giving bounded outputs. During field experiments we set $\sigma_\vartheta = 0$ as this tends to yield better performance and smoother outputs [34].

III. METHOD

The control objective of the RL controller is to control the attitude of the aircraft to the desired reference attitude. We use standard aircraft nomenclature and coordinate systems [26], as well as a roll-pitch-yaw Euler angle parameterization of attitude. The heading/yaw angle is typically not controlled directly, but rather through banked-turn maneuvers [26]. Therefore, the natural choice of controlled states are the roll angle ϕ, and the pitch angle θ. We assume that the UAV is equipped with control surfaces such that the roll and pitch angles are controllable (an assumption that is satisfied by most UAV designs). The Skywalker X8 seen in Fig. 1 is used in our field experiments. It has two elevon control surfaces, one on each wing, which can be driven together to produce a pitching moment, or driven differentially to produce a rolling moment. In addition, it has a propeller that can produce a thrust force along the longitudinal axis of the UAV. In the simulation environment, a PI-controller is used to control airspeed using the propeller throttle [12].

As a general approach, we tested new ideas in the simulation environment and made extensive use of sim-to-sim experiments where we studied how the controller transferred from simulation with one set of parameters to a simulation with another set of parameters. We then tested the most promising controllers in flight experiments in the field and adjusted our approach based on the insight we gathered from the flight experiments. The simulation environment software is made open-source and is available at [35].

A. Controller Architecture and State Design

We identified in our previous work that limiting the state vector to only the most useful information and reducing redundancy is important for the rate of convergence, and to prevent the controller from learning spurious relationships. This has also been observed in other research [17]. At every time step we measure the following information:

$$m_t = [p_t, q_t, r_t, \alpha_t, \beta_t, V_{a,t}, \delta_{r,t-1}, \delta_{l,t-1}, \\ I_{\phi,t}, I_{\theta,t}, \phi_t, \theta_t, e_{\phi,t}, e_{\theta,t}]^\top \tag{6}$$

$$I_{*,t} = \gamma^I I_{*,t-1} + e_{*,t}, \; \gamma^I = 0.99, \; I_0 = 0, * \in \{\phi, \theta\} \tag{7}$$

where t is the time index, $\omega_t = [p_t, q_t, r_t]^\top$ is the angular velocity in the body-fixed frame, α_t is the angle-of-attack, β_t is the sideslip-angle, $V_{a,t}$ is the airspeed, $\delta_{\{r,l\},t-1}$ represent the previous output of the RL controller, in this case the commanded deflection angles of the right and left elevons, $e_{*,t} = *_t - *_{r,t}$ is the state tracking error where subscript r

denotes the state reference, I_* is the integrator of the state error and γ^I is the integrator decay rate. The integration decay scheme follows [19], and facilitates boundedness of the integrator state. Lastly, because NNs are known to converge faster given normalized inputs, the measurements are normalized using running estimates of mean and variance for each component before being fed to the controller.

Due to unmeasured effects such as turbulence and the sim-to-real measures described in Section III-D, the attitude control problem is partially observable. Furthermore, to enable the controller to adapt to the dynamics of the field experiments, we wish to enhance the controller with the capability of inferring the dynamics around the current state. A common approach to achieve this effect is to use a recurrent neural network (RNN) [23, 36]. However, we found that using a one-dimensional convolution over the time dimension as the input layer yielded similar control performance, and therefore prefer it since it is significantly less complex than the RNN. We therefore include the h last measurements (6) in the state vector, where $\hat{m}_t$ indicates a noisy measurement to be defined in Section III-D:

$$s_t = [\hat{m}_t, \hat{m}_{t-1}, \ldots, \hat{m}_{t-h}]^\top \tag{8}$$

$$[\delta_{r,t}, \delta_{l,t}]^\top = \pi_\vartheta(s_t) + [\delta_{r,\text{trim}}, \delta_{l,\text{trim}}]^\top \tag{9}$$

such that the total size of the state vector is $|s_t| = |m_t| \cdot h$. The convolutional input layer scales favorably in number of learned parameters compared to a fully-connected (FC) layer: it scales linearly in $|m_t|$ as opposed to multiplicative for the FC layer, and it is constant for h. The convolutional layers' output size is $F \cdot |m_t|$ where F is the number of learned convolutional filters, and each filter has size h. The memory capacity of the state vector can therefore be increased as required to give sufficient history to infer the dynamics, with only a slight increase in the number of parameters. Through a small grid search using rate of learning and asymptotic performance as metrics we found $F = 8$ and $h = 10$ to work well. The complete RL controller architecture is shown in Fig. 2.

The output of the RL controller is the commanded states of the controlled system's actuators relative to the trim-point (9). The nominal elevon deflection angles $\delta_{r,\text{trim}} = \delta_{l,\text{trim}} = 0.045$ are calculated using a standard trim routine for horizontal, wings-level flight based on the model in Section III-C [26]. The target UAV for the field experiments, the Skywalker X8, has elevon actuators and we therefore chose to have the controller output the desired deflection angles of these directly, in order to provide RL with as direct control as possible. This choice is fairly arbitrary, however, and experiments showed that outputting virtual elevator and aileron angles (the sum and difference, respectively, of the elevon angles defined by (27)-(28)) instead yield similar performance.

B. Reward and Objective Design

We found sparse rewards to yield better results than shaped rewards, both in terms of rate of learning and in terms of asymptotic performance. A sparse reward is one that is nonzero only for some subset of the state space. It has the benefit that it is easier to formulate than hand-crafted
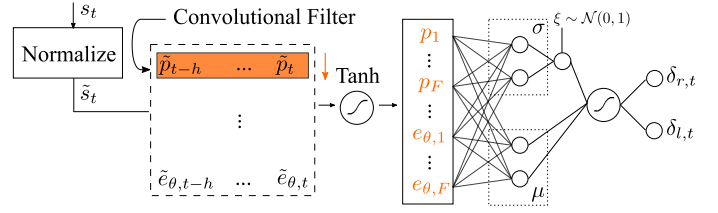

Fig. 2. Architecture of the RL controller, superscript $\sim$ signifies normalized states.

shaped rewards, and would therefore be more transferable to other UAVs with fewer adjustments necessary. The reward is formulated as follows:

$$R(s_t, a_t) = \lambda^\phi B(e_{\phi,t}) + \lambda^\theta B(e_{\theta,t}) + \lambda^{\dot{\phi}} B(\dot{\phi}_t) + \lambda^{\dot{\theta}} B(\dot{\theta}_t) \tag{10}$$

$$B(\cdot) = \begin{cases} 1 & \text{if } |\cdot| \le \cdot^b \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$e_{\phi_t}^b = 3°, \ e_{\theta_t}^b = 3°, \ \dot{\phi}_t^b = 4.3°/s, \ \dot{\theta}_t^b = 4.3°/s \tag{12}$$

$$\lambda^\phi = 0.5, \ \lambda^\theta = 0.5, \ \lambda^{\dot{\phi}} = 0.167, \ \lambda^{\dot{\theta}} = 0.167 \tag{13}$$

where superscript b refers to the goal-bound on the variable and the λs are weighting factors. This reward ensures that the controller tracks the setpoints with accuracy as specified by the bound, while the rewards on the derivatives of the controlled states discourage high rates. Our method is not very sensitive to the size of the bound, but generally larger bounds accelerate learning at the expense of tracking accuracy.

When transferring from a simulator environment to the field, it is important to consider how the actuation system impacts the effects of actions. That is, while high-gain bang-bang control may be an optimal strategy in the simulator, frequently changing the setpoints of the actuators introduces considerable wear due to the high currents generated as a result of the switching. In our previous work [12] (and indeed in other works [13]) this problem is addressed through a term in the reward that discourages changing the setpoints. We now take a different approach to this problem, using the conditioning for action policy smoothness (CAPS) method [37]:

$$J^{\text{TS}}(\pi_\vartheta) = ||\pi_\vartheta(s_t) - \pi_\vartheta(s_{t+1})||_2 \tag{14}$$

$$J^{\text{SS}}(\pi_\vartheta) = ||\pi_\vartheta(s_t) - \pi_\vartheta(\hat{s}_t)||_2, \quad \hat{s}_t \sim \mathcal{N}(s_t, 0.01) \tag{15}$$

This method adds two regularization terms to the loss, a temporal smoothness term (14) and a spatial smoothness term (15). As the authors demonstrate, this method is more successful in generating controllers that yield smooth control signals compared to the action reward-term approach. Additionally, removing the action term from the reward simplifies the problem of learning the action-value function since the reward now contains fewer disparate parts, thereby accelerating learning. Instead, the gradient ascent scheme calculating the parameter updates is conditioned towards policies that are smooth in

the output. Finally, we add a regularization term on the pre-activation $\pi_\vartheta^{\mathrm{PA}}$ (that is, before applying the hyperbolic tangent in (5)) of the output:

$$J^{\mathrm{PA}}(\pi_\vartheta) = ||\pi_\vartheta^{\mathrm{PA}}(s_t)||_2 \qquad (16)$$

This helps in reducing the gain of the controller, especially for small errors, as it essentially tells the controller that it needs to have a strong benefit to move the actuators away from the trim-point. Additionally, we find it accelerates learning as the controller is biased towards non-aggressive control, which in conjunction with hindsight experience replay (HER) means the controller quickly discovers how to achieve the sparse stabilizing objective. Thus, the objective we optimize is defined as:

$$J(\pi_\vartheta) = J^{\mathrm{SAC}}(\pi_\vartheta) + \lambda^{\mathrm{TS}} J^{\mathrm{TS}}(\pi_\vartheta) + \lambda^{\mathrm{SS}} J^{\mathrm{SS}}(\pi_\vartheta) + \\ \lambda^{\mathrm{PA}} J^{\mathrm{PA}}(\pi_\vartheta) \qquad (17)$$

$$\lambda^{\mathrm{TS}} = 5 \cdot 10^{-2}, \ \ \lambda^{\mathrm{SS}} = 10^{-1}, \ \ \lambda^{\mathrm{PA}} = 10^{-4} \qquad (18)$$

C. UAV Model

For the simulated environment, we use a model of the Skywalker X8 UAV based on previous modeling efforts [38, 39, 40, 41]. The model structure is standard in the literature [26, 42] and is based on a single rigid body Newton-Euler formulation affected by forces and moments due to gravity, aerodynamics, and propulsion effects. An estimate of the inertia matrix is provided in [41] based on bifilar pendulum experiments. Results from wind-tunnel experiments are provided in [38] for aerodynamic coefficients, and in [39] for the propulsion system model. This data is complemented by computational fluid dynamics (CFD) simulations from [38, 40]. For a more detailed description of the simulation model, see our previous work in [12].

We collected a short data series to assess the quality of the dynamic model. To excite the system dynamics, we used the actuator signals from the baseline attitude controller and perturbed them with chirp signals before mapping them to the elevon deflections. The start and end frequencies of the chirp signals were $8\,\mathrm{Hz}$ and $12\,\mathrm{Hz}$, respectively. A dynamic mode analysis of the model indicates that this is the dominant frequency spectrum of the X8. The signal duration was 15 seconds and we used a peak-to-peak amplitude of 20 degrees.

The aerodynamic coefficients that are calculated based on recorded sensor data and the inertia matrix of the vehicle are shown in Fig. 3. Following [26], the coefficient subscript L, D, Y, l, m, n denotes lift, drag, side force, roll moment, pitch moment and yaw moment, respectively. These results show that despite the modeling efforts, there are still significant discrepancies between the predicted and measured data, particularly in the pitching moment coefficient, C_m.

D. Sim-to-Real Measures

The main sim-to-real measure employed in the method is domain randomization. As shown in Section III-C, there is a significant reality gap, and as such we want to avoid the
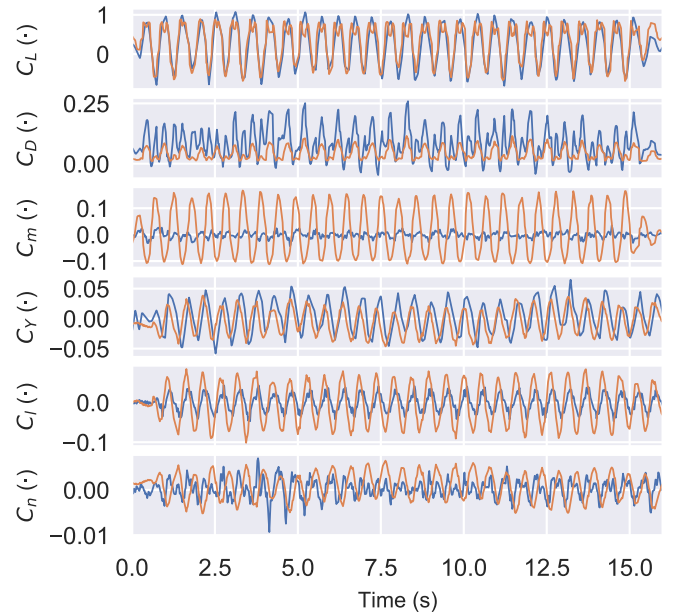

Fig. 3. The aerodynamic coefficients of the UAV in a longitudinal (top three) and a lateral (bottom three) chirp signal test sequence for elevator and aileron, respectively, based on IMU data (blue) and model prediction (orange).

RL controller overfitting to the simulation environment. The intuition behind the domain randomization technique is that learning over a distribution of possible UAV models should robustify the controller. To this end, we assess the uncertainty in the estimate of every parameter of the UAV model and use this uncertainty to construct a probability distribution over its range of probable values (see [35] for details). The coefficients of the rate-dependent terms of the UAV model have larger sampling ranges since these are not estimated based on wind tunnel data but rather on uncertain CFD simulations [38]. The sampled values are also clipped as indicated to avoid extreme unrealistic values. At the start of every episode, we sample a value for each parameter from its distribution, together constituting one realization of the UAV model.

The UAV sensor suite is subject to noise in its measurements. To model these, we first estimated the real hardware's noise characteristics, then we emulated this in the simulator environment. We model the measurement noise as an Ornstein-Uhlenbeck (OU) process (19), which in addition to white noise gives rise to effects like measurement drift:

$$\hat{m}_t = m_t + w_t, \ \ w_t \sim OU(\mu_m, \sigma_m, \theta_m), \ \ \mu_m = 0, \ \ \theta_m = 1$$
$$\sigma_m = 0.005 \cdot [1.5, 1.5, 1.5, 1, 1, 15, 0, 0, 0, 0, 1, 1, 0, 0]^\top \quad (19)$$

where $\mu_m, \sigma_m, \theta_m$ are the mean, variance, and rate of mean-reversion parameters of the measurement noise. Note that we do not add noise to the error and integrator states, as these are already affected by the noise in the measurement of the state that is used to calculate the error, while the previous output of the controller is by nature free of noise. The sensor suite has an update rate of 50Hz, and we therefore chose this as the control frequency as well. In the simulation environment we

add exponentially distributed noise on top of the fixed control frequency in order to simulate sensor timing-variability:

$$\Delta_t = \Delta_0 + z_t, \;\; z_t \sim \mathrm{Exp}(\kappa), \;\; \kappa \sim \mathcal{U}(250, 1000) \quad (20)$$

where Δ_t is the simulation step size at step t, $\Delta_0 = 0.02\,\mathrm{s}$ is the base control frequency, and z_t is exponentially distributed noise whose parameter κ is drawn uniformly at the start of every episode.

Although not strictly sim-to-real measures, the adjustments to the RL objective described in Section III-B in the form of CAPS and the pre-activation term also serve to improve the transferability from simulation to reality, as they encourage less aggressive lower-gain control. Another major effect present in the field is atmospheric disturbances such as wind and turbulence. We model turbulence with the Dryden turbulence model [26], and a steady wind component whose direction and magnitude between $0\,\mathrm{ms}^{-1}$ and $15\,\mathrm{ms}^{-1}$ is sampled at the start of each episode. The last effect we found was highly impactful for successful transfer was the actuation delay, i.e. the time it takes before the output of the controller is applied to the system, a result which was also found in [23]. The simulator contains a constant actuation delay of $100\,\mathrm{ms}$, while we believe this is a significant overestimation of the delay of the real system, we motivate this choice in Section V-B.

E. Simulator Episode Design

The standard design of episodes for UAV control in the literature seems to be short episodes with a single constant desired attitude [13, 17]. We found that having constant setpoints accelerates learning, however, the operation of the UAV in the field typically sees the navigation system continuously update the desired attitude. To align these considerations, we employ fairly long episodes of length 900 steps ($18\,\mathrm{s}$) where setpoints are kept constant but resampled every 150 steps. To ensure sufficient diversity of the state trajectories and transitions used to update the parameters of the RL controller, we sample initial conditions as shown in Table I. Considering that the main objective of the simulation environment is to ready the controller for the field, we sample initial conditions mostly from the linear region of the model, as this is where the UAV model is assumed to have the most validity. Note that while the range of initial conditions is somewhat limited, there is nothing stopping the controller from exploring the full state space. Furthermore, since the initial states of the actuators are also randomized the sampled initial conditions could cause instability, such that the controller must learn to recover.

F. Reinforcement Learning Algorithm

To develop the RL controller we use the SAC algorithm and augment the collected data using HER [43], see [35] for hyperparameters and code. We chose the SAC algorithm because it is off-policy, and therefore has comparatively high data efficiency among RL methods, and furthermore the policy is explicitly trained to handle perturbations from the inherent randomness, which tends to yield more robust policies that

TABLE I
INITIAL CONDITIONS FOR THE EPISODES ARE UNIFORMLY SAMPLED
FROM THE INDICATED RANGES.

Variable	Range	Unit	Variable	Range	Unit
ϕ	-40, 40	degrees	ϕ_r	-60, 60	degrees
θ	-15, 15	degrees	θ_r	-25, 20	degrees
V_a	13, 26	m/s	α	-8, 8	degrees
ω	-60, 60	degrees	β	-10, 10	degrees
$\delta_{r,l}$	-30, 30	degrees			

transfer better than non-entropy-regularized algorithms. Note that we employ the technique of initializing the replay buffer of the algorithm with 5k data samples (corresponding to 100 seconds of flight at 50Hz), which is a common technique in RL to help the policy with the initial exploration phase. This data is entirely independent of the learning controller being trained and is obtained by uniformly sampling random actions from the set of possible actions in the simulator environment. This data could also stem from other sources such as historical data gathered by a human pilot or another controller, which might be more suitable when performing exploration exclusively in the field. Since this data is independent of the learning controller, we do not count it towards the data requirement of our method and do not include it in the learning curve graphs.

G. Experimental Platform

Our custom avionics stack is based on the low-level control architecture developed at the NTNU UAV-lab [33]. It consists of a Cube Orange flight controller running the (industry standard) ArduPlane open-source autopilot [44], and a Raspberry Pi 4 running the DUNE Uniform Navigation Environment [45]. During experiments, the total flight weight of the Skywalker X8 is $3.8\,\mathrm{kg}$.

The RL controller is implemented as a DUNE task in C++ with the NN implemented in TensorFlow. Sensor data and state estimates from ArduPlane are sent through a serial connection to the Raspberry Pi, providing all necessary data for the RL controller. The NN controller output is converted to PWM duty cycle and sent to the elevon servos using a PCA9685 servo driver, interfaced through I2C from the Raspberry Pi. A PWM multiplexer supports switching between the RL controller output and ArduPlane. This enables the pilot to always take control during testing, either through manual controls, or through ArduPlane's standard autopilot. This switching mechanism enables us to safely engage the RL controller in flight, while takeoff and landing are performed by the pilot operating our tried and tested avionics stack [46].

IV. EXPERIMENTAL RESULTS

This section presents the main experimental results, collected during two days of flight experiments at Agdenes Airfield, Breivika, Norway in September 2021. During the first day, we enjoyed calm weather and perfect flight conditions, with a mean wind speed (as estimated by ArduPlane's Kalman Filter) of less than $4\,\mathrm{ms}^{-1}$. The second day of flight

tests, however, presented challenging weather conditions, with frequent gusts and a mean wind speed of approximately $12.5\,\mathrm{m\,s^{-1}}$ (70% of the Skywalker X8's cruise speed of $18\,\mathrm{m\,s^{-1}}$).

We present three types of data, differing mainly by how roll and pitch angle references are provided:

1) References are given by the pilot, mimicking Ardu-Plane's fly-by-wire-A (fbwa) mode (Section IV-A).
2) References are provided by ArduPlane's guidance system, which is set to track a rectangular waypoint pattern (Section IV-B).
3) References are set by a predefined, automated series of steps (Section IV-C). Similar maneuvers are also performed with an implementation of the ArduPlane PID attitude controller, with the response compared to that of the RL controller.

In contrast to the training phase, where the throttle actuator used to control airspeed is operated by a PI controller (see [12] for details), the throttle is either under manual control by the pilot (fbwa) or controlled by ArduPlane (auto/steps). In figures presenting flight results, the dashed orange line corresponds to state reference, while in the elevon plots, the blue and orange lines correspond to the right and left elevon, respectively, with minimum and maximum deflections of $-30°$ and $+30°$.

A. FBWA Mode

Fig. 4 shows an excerpt from the flight experiments where a human pilot decides the desired attitude of the UAV. The RL controller is able to closely track the desired attitude even for the most difficult and aggressive maneuvers, while producing smooth outputs for the actuators. We do however note a consistent steady-state error. Towards the end of the maneuver, we observe that the roll response is non-symmetric, that is, rolling to the left (towards negative roll angles) is slower than rolling in the opposite direction. We investigate and discuss this matter, as well as the steady-state error, in Section V.
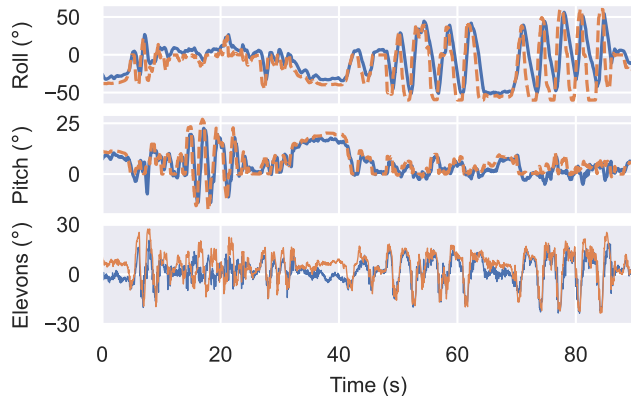

Fig. 4. Fbwa mode using references (dashed orange line) from the pilot, showing the attitude states and right (blue) and left (orange) elevon signals.

B. Auto mode

Fig. 5 shows the results for the RL controller operating with references provided from the ArduPlane guidance system, set
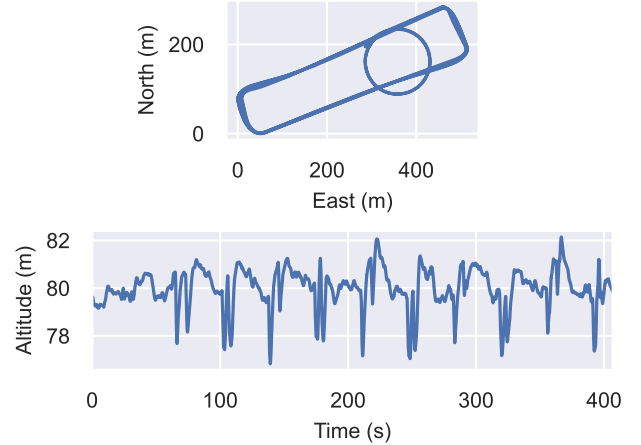

Fig. 5. Position plot showing how the RL controller is able to take references from ArduPlane's guidance system in Auto mode, and effectively follow prescribed paths. First, a loiter, then a square waypoint pattern.

to track a square waypoint pattern. Before tracking the square, the UAV loiters in a circular pattern for a while. Despite some steady-state offset, especially for the roll angle error, the UAV successfully completes the specified mission. This is because the outer-loop guidance controller can compensate for this error, and still achieve convergence when faced with disturbances such as wind and offset in inner-loop control. This is similar to how a pilot supplying manual references would offset the references to keep the intended course.

During turns, a certain altitude drop is seen from the right part of Fig. 5. This is caused by the aggressive turn radius accompanied by drops in the pitch angle. This effect can be reduced by tuning the guidance system to be less aggressive (e.g. by increasing the turn radius) or reducing the maximum allowable roll angle setpoint, which is set to be 55 degrees. A similar drop in pitch angle is also seen when using the default ArduPlane attitude controller.

C. Step sequences and Comparison with ArduPlane PIDs

Step responses for the RL controller, as well as the Ardu-Plane PID controller, are displayed in Fig. 6. The RL controller shows comparable transient performance to that of ArduPlane, the main difference being the steady-state error of the RL controller. Additionally, the pitch response of the PID controller is slightly more aggressive. However, this could be changed by tuning the controller.

The control signals generated by the RL controller are relatively smooth and well-behaved but include some high-frequency components not seen in the PID response. Apart from that, the control input looks qualitatively similar, with similar magnitude. For a quantitative comparison we employ the smoothness metric defined in [37] which jointly considers the amplitudes and frequencies of the control signals:

$$Sm = \frac{2}{nf_s} \sum_{i=1}^{n} M_i f_i \qquad (21)$$

where M_i is the amplitude of the i'th frequency component f_i, and f_s is the sampling frequency. On this metric the PID

measures at $6.20 \cdot 10^{-4}, 6.30 \cdot 10^{-4}$ for the roll and pitch maneuvers in Fig. 6, respectively, while RL measures 2% and 44% higher at $6.30 \cdot 10^{-4}, 9.05 \cdot 10^{-4}$. This metric shows that RL has comparable smoothness in its output with the PID controller, but also indicates the higher frequency components of the RL controller's output in Fig. 6. It is not clear why there is such a discrepancy between the two maneuvers for the RL controller, but this data is as mentioned subjected to considerable turbulence and wind, and could therefore be caused by transient gusts.

While the former results were gathered on a calm day with virtually no wind, these maneuvers are executed in harsh wind conditions on day two. The UAV also suffered structural damage (not while under RL control) after collecting the PID data, before redoing the experiments with RL. The vehicle had to be repaired with a new wing and duct tape, causing a change in the UAV's trim point. Thus, the presented results demonstrate the RL controller's robustness towards model mismatch and varying wind conditions, including heavy gusts.

To achieve a fair comparison, the ArduPlane PID is implemented in the same software stack and ran with the same hardware architecture as the RL controller (described in Section III-G). In particular, this means that any increased signal latency introduced in our setup does not affect the comparison.

V. DISCUSSION

The experimental results of Section IV show that the RL controller performs well compared to a state-of-the-art open-source autopilot, and is robust to disturbances caused by harsh wind conditions. The control performance of the RL controller across the various flight modes speaks to its ability to generalize further than just the maneuvers encountered during training. In particular, no guidance controller was present during training.

Despite the promising results, there is room for improvement. In this section, we further discuss how performance can be improved, the iterative development process, and training, and we perform a linear analysis to gain further insight into the behavior of the RL controller.

As noted in Section IV-A, the roll response of the RL controller is non-symmetric, meaning that rolling towards the left wing is slower than rolling to the right. This is supported by the pilot's qualitative assessment during flight. We found that this effect was caused by an overestimation of the simulated propeller torque, which was found to be less prominent on the physical UAV than expected, presumably due to the mechanical mounting of the propeller. This causes a bias in the RL controller, which has learned to counteract the propeller torque. To remedy this, we trained a new RL controller in a simulation model without propeller torque, which was briefly tested in the field to verify our hypothesis. The new controller exhibited a more symmetric roll response, as expected.

A. Steady-State Errors

We experimented with several techniques in order to address the steady-state error of the RL controller observed in flight experiments: pure integrator (no decay), higher decay factor (e.g.

0.999), having integration separate from the NN controller with learned integration gains, shaped rewards, and training with input disturbances. Whereas some of these measures reduced the steady-state error to some degree, none were successful in entirely eliminating it.

We note that there is no consistent steady-state error in the simulator in the same way we observe in field experiments, i.e. consistently over or under the reference with a consistent magnitude. The controller has learned to use integral action to reduce steady-state error from disturbances in the simulator, but not in a way that transfers to the field. This could be because the controller is overfitting to the simulator, thus the larger tracking errors in the field combined with the hyperbolic tangent saturating functions of the NN causes the integrator states' effect on the output to saturate prematurely.

An effective way to address this problem is to estimate the steady-state error and then add the estimated value to the references provided to the RL controller, as was done in the flight experiment shown in Fig. 7. As can be seen, this simple technique can fully compensate for the steady-state error and may also be automated using an integrator in an outer loop to estimate the steady-state error [47]. Furthermore, there are compelling arguments for not having integral action in the inner-loop attitude controller, as adding integral action to the controller necessarily reduces the phase margins and the achievable bandwidth [26].

B. Oscillations: Illustration of Iterative Development

Initial field experiments were characterized by excessive oscillations in the attitude response of the UAV, especially in pitch, necessitating halving the RL controller's outputs in order to keep the aircraft airborne. These oscillations were not present in the simulator, as such, we suspected that this was (at least in part) caused by the simulator overestimating the natural damping present in the aircraft. Therefore, we reduced the C_{m_q} (pitch damping) parameter by a factor of 10. While this reduced the oscillations somewhat, there were still significant oscillations in the response, see Fig. 8.

We estimated a typical actuation latency for the system of $10 \, \mathrm{ms}^{-1}$. In sim-to-sim experiments, where we raised the latency of the control system during the exploitation phase of a controller trained with $10 \, \mathrm{ms}^{-1}$ latency, we observed similar oscillatory responses as in the flight experiments and noted its relationship with increasing latency. We then trained an RL controller where the latency was set to $100 \, \mathrm{ms}^{-1}$ during the learning phase. This controller trained with higher latency almost entirely eliminated the oscillations to the levels shown in the field experiment figures. Favoring robust controller design, we increased the base latency of the simulation environment to $100 \, \mathrm{ms}^{-1}$, even though we believe that this is a significant overestimation of the true latency of the real system.

C. Linear Analysis

In order to better understand how the RL attitude controller operates, we analyze its sensitivity to the input variables. In Fig. 9 we have plotted the open-loop response of the controller as a function of a single perturbed input. The rest of the state
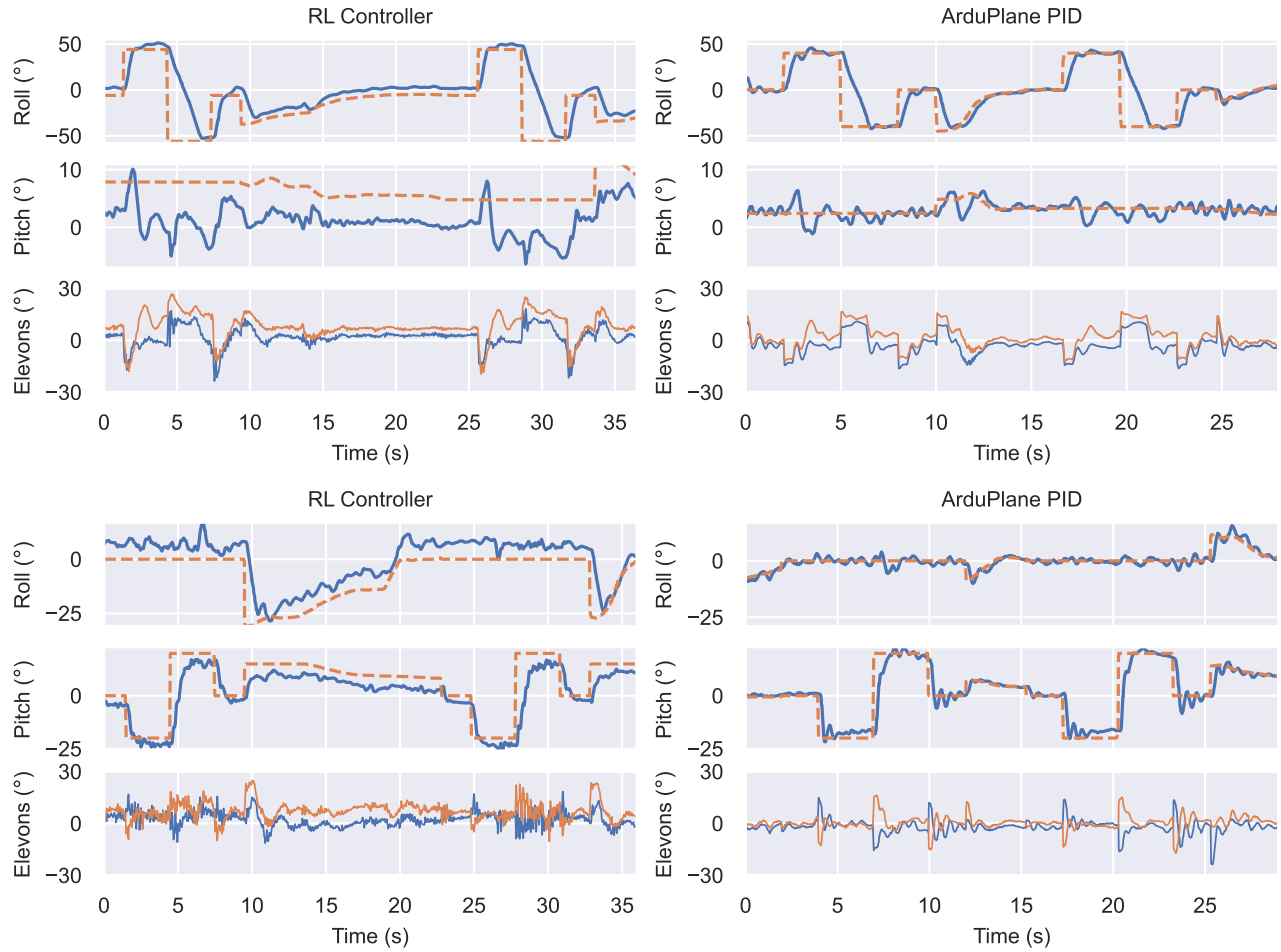
Fig. 6. Comparison between the RL controller and the ArduPlane PID controller for steps in the roll (top) and pitch (bottom) references (dashed orange line).
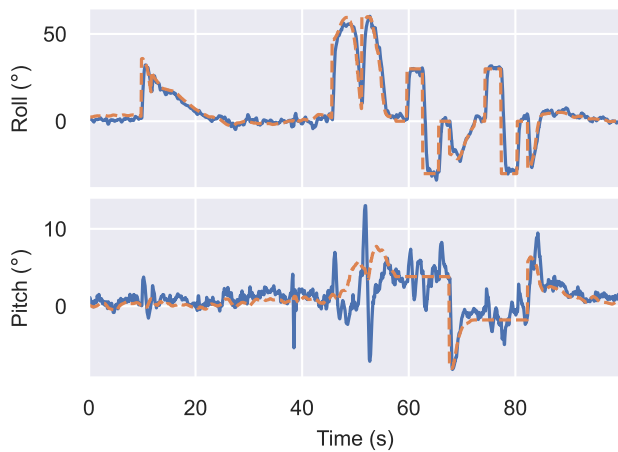

Fig. 7. Response of the RL controller where the steady-state error has been estimated and references (dashed orange line) adjusted to compensate.
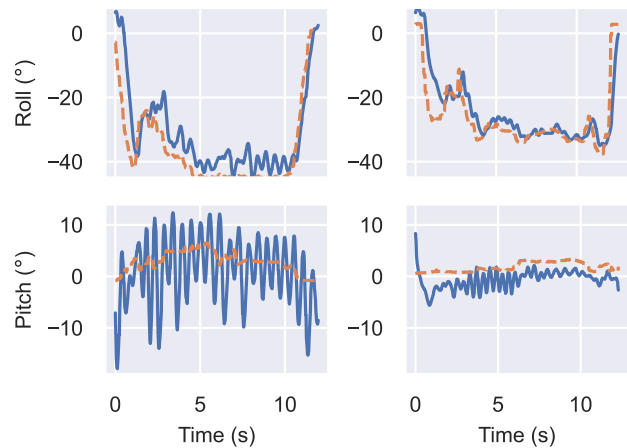

Fig. 8. Oscillatory attitude response of initial flight experiments, for one controller trained with original (left) and one with reduced (right) C_{m_q}.

vector is kept constant at the steady-flight value, i.e. zero for all variables except the airspeed V_a which is set to the cruising speed of $18\,\mathrm{m\,s^{-1}}$, and the angle-of-attack α and pitch angle θ, which are kept at the trim values necessary to generate lift for level flight, while the input values for previous time steps are kept constant in the time dimension. To be able to compare

the results with ArduPlane, we translate the elevon outputs into virtual elevator and aileron commands using the inverse of (27)-(28). The figures and tables are presented in terms of these virtual commands, which also have a more intuitive and straightforward effect on the roll and pitch angles.

TABLE II
LINEARLY APPROXIMATED GAINS AT LEVEL-FLIGHT, WHERE EACH INPUT IS PERTURBED IN ISOLATION.

Controller	$\frac{\partial \delta_a}{\partial e_\phi}$	$\frac{\partial \delta_e}{\partial e_\theta}$	$\frac{\partial \delta_a}{\partial I_\phi}$	$\frac{\partial \delta_e}{\partial I_\theta}$	$\frac{\partial \delta_a}{\partial p}$	$\frac{\partial \delta_e}{\partial q}$
RL	1.268	-3.320	-0.005	0.006	-0.008	0.223
PID	1.630	-1.081	0.052	-0.052	-0.024	0.031

The saturating effect of the hyperbolic tangent nonlinearity on the RL controller is distinctly present in the responses. This is a desired effect as we know that any input should have a bounded effect on the output, which gives robustness towards possible measurement errors or misalignment of the dynamics of the simulation and real environments. The controller makes use of all its inputs, with the previous outputs of the controller having the most significance for the current output (the typical values for most states in Fig. 9 are close to the level-flight value in the center and will thus use a limited range of the response curve, while the previous output of the controller frequently employs the full range). This makes sense as the controller is conditioned towards smooth outputs, as described in Section III-B, which means that a reasonable initial guess of any action is to be similar to the previous action. Moreover, the fact that the previous output (left and right elevons) do not have a symmetric effect on the subsequent output (there is no mechanism enforcing symmetry in the learning controller) could be a motivating factor to instead employ (virtual) elevator and aileron as outputs of the RL controller.

Since the control authority of the actuators increases with airspeed, we investigated if the RL controller has learned to scale its outputs depending on airspeed, an effect that is included in the ArduPlane controller. With no further documentation (due to space constraints), we state that this is not the case. However, it has learned to bias the response, essentially shifting the curves in Fig. 9 up, as airspeed increases in order to compensate for the change in trim-point with airspeed.

To estimate sensitivities wrt. an input we take a linear approximation of its response curve by using the slope of the tangent line at level-flight conditions. The result is shown in Table II, and compared to the ArduPlane PID controller gains (see Appendix A). The RL controller is noticeably more aggressive in the pitch error, while simultaneously introducing more damping through the angular velocity component q. This is evident in Fig. 6 where the RL controller exhibits less oscillation in the pitch response. The estimated gains for the integrator states in Table II are not representative of the response curves for these states, as the response curve exhibits cubic characteristics with a small opposing region around the level-flight value. Thus, for these states, a linear approximation over a larger region would be more descriptive. Overall, the gains of the RL controller are similar to those of the PID controller, which increases the trust in the RL controller. On the other hand, the dynamic aspect caused by integral states and data from previous time steps increases the complexity of the analysis and thus limits the conclusions that can be drawn from it.
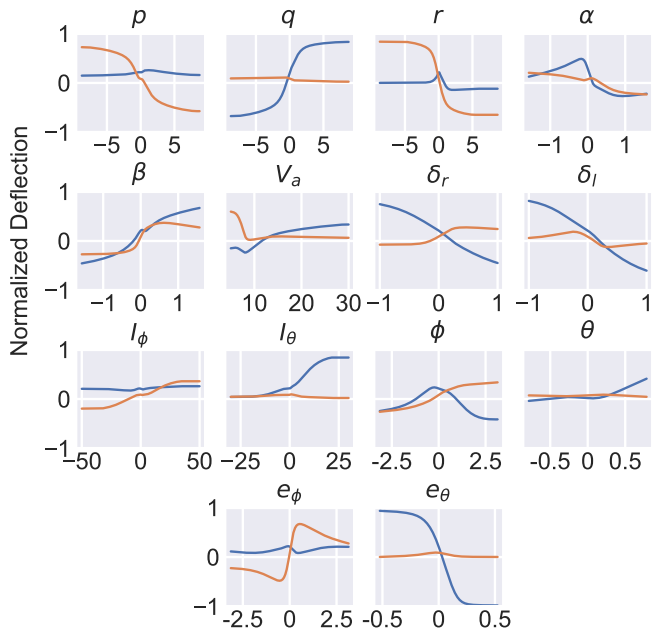

Fig. 9. Open-loop level-flight response of the RL controller when perturbing one input at a time. The x-axis is in the units of the corresponding state. The lines are elevon outputs mapped to aileron (orange) and elevator (blue).
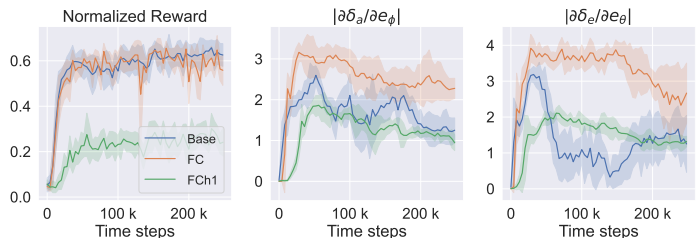
D. Data Requirements

Fig. 10. The learning phase of the proposed RL controller, showing normalized mean episode reward and error-proportional gains. The solid line represents a rolling average mean value while the shaded region represents one standard deviation over three randomly seeded controllers. Base refers to the method as presented in Section III, the FC version replaces the convolutional layer with an FC layer, and the h1 version has no history in the input.

In this section, we attempt to quantify the data efficiency of our method. We start by presenting the learning phase of the controller in the simulation environment (Fig. 10), demonstrating that our method produces proficient controllers with a number of data samples on the order of 10s of thousands, and then demonstrate that the learning controllers are flight-worthy in the field after experiencing only three minutes of real-time flight data (Fig. 11). It is difficult to compare this result directly to the existing literature, for reasons outlined in the literature review of Section I, that is, no other reported work study the full attitude control problem of fixed-wing UAVs using RL. Other works study either a limited version of the attitude control problem, or consider other aircraft designs (e.g. quadcopters) whose dynamics are more linear and controllable than the dynamics of a fixed-wing UAV. Nonetheless, works in the existing literature report a data requirement on the order
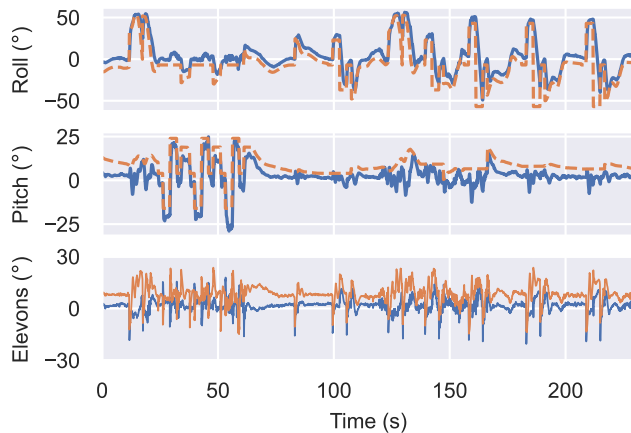
Fig. 11. An RL controller that has trained for only 3 minutes of real-time flight in the simulation environment is airworthy, able to track references (dashed orange line) reasonably well.

of 100s of thousands or millions of data samples, meaning our result presents a significant improvement in terms of data efficiency. Further, we would argue that this improvement in data efficiency is significant because it suggests that learning the controller entirely in the real world (i.e. no simulation required) is possible. Collecting data for a few minutes of flight seems reasonably achievable through some form of safe exploration or guided learning where, for instance, the learning controller is monitored by a pilot or other proficient controller that can assume control should dangerous situations arise.

The evolution of the learning phase for the RL controller as a function of time steps is shown in Fig. 10. For every version in Fig. 10, we train three controllers each with a different initial random seed and average the results over the controllers. The rewards are normalized so that 1 corresponds to attaining the maximum reward as defined in (10) at every step (although this is not physically achievable) and 0 corresponds to obtaining no rewards at all. Base refers to the RL controller as presented in Section III that was used in the field experiments. To assess the contribution of the convolutional input layer, we trained one version where the input layer is replaced with an FC layer, and further to test the importance of the history of states in the state vector we train one model with an FC input layer and with $h = 1$ (labeled FCh1).

The base version learns fast, reaching convergent performance after around 40k time steps. This corresponds to about 13 minutes of real-time flight or 16 minutes of wall-clock training time when trained entirely on an i7-9700 Intel CPU with a single data-generating-agent in the simulator[1]. We verified in the field that the RL controller is flightworthy long before this point: The controller shown in Figure 11 exhibits decent and stable performance (although unable to follow the most aggressive maneuvers) after only 10k time steps of training in the simulator (corresponding to three minutes of real-time flight). We conjecture that the data efficiency of our method limits overfitting to the simulation

environment and therefore transfers better to the field, although this statement requires further evidence such as a bench of learning controllers with varying time in simulation to verify.

We find that our training method is stable in the sense that the performance differences between controllers with different seeds are small, within a few percent. The FCh1 version without state history is never able to learn to consistently stabilize the UAV at the desired attitude in the time frames we considered. The FC version with state history on the other hand achieves comparable rewards to the base version, showing the importance of history in the RL input state. The base version reaches peak performance slightly faster than the FC version, and further its proportional gains are considerably lower. This is also evidenced by the smoothness metric (21) for which the base version scores 50% lower than the FC version. The lower gains and the smoothness metric indicate that the convolutional input layer provides a superior ability to predict the system response and thus provide smoother responses in attitude and control signals, while the FC version is more reactive and oscillatory.

VI. CONCLUSION

This paper has presented a data-efficient method for learning attitude controllers for fixed wing UAVs using RL. The learning controller is able to operate directly on the nonlinear dynamics, and therefore could extend the flight envelope and capabilities of autopilots. The high data efficiency of the presented method facilitates transfer to control of the real UAV by limiting overfitting to the simulated model. We demonstrate that the learned controller has comparable performance to the existing state-of-the-art ArduPlane PID autopilot, and is capable of tracking prescribed paths from a guidance system while generating smooth actuation signals and attitude responses. Key factors behind the success of the method were robustifying the controller through increasing its phase margins by learning with significant actuation delay and diversifying the simulated dynamics, as well as incentivizing non-aggressive control through sparse rewards and additional objective terms enforcing temporal and spatial smoothness in the controller outputs.

Further work in this direction should investigate if RL solutions to more complex flight control problems also transfer well to the field, e.g. deep-stall landings or end-to-end path following. The problem of limited integral action should also be further investigated. Moreover, learning from real data, be it historical or generated online by the learning controller, is an intriguing research direction that would alleviate the need for accurate nonlinear models. The data efficiency of our method shows that this should in fact be feasible.

---

[1]Note that the wall-clock training time can essentially be made arbitrarily short with parallelization of more agents and more powerful computing hardware

REFERENCES

[1] E. Bulka and M. Nahon, "Automatic control for aerobatic maneuvering of agile fixed-wing uavs," *J Intell Robot Syst*, vol. 93, pp. 85–100, 2019.

[2] R. Cory and R. Tedrake, "Experiments in fixed-wing uav perching," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008.

[3] S. Mathisen, K. Gryte, S. Gros, and T. A. Johansen, "Precision deep-stall landing of fixed-wing uavs using nonlinear model predictive control," *J Intell Robot Syst*, vol. 101, no. 24, 2021.

[4] T. Cunis, D. Liao-McPherson, I. Kolmanovsky, and L. Burlion, "Model-predictive spiral and spin upset recovery control for the generic transport model simulation," in *2020 IEEE Conference on Control Technology and Applications (CCTA)*, 2020, pp. 1–7.

[5] C. Liu and W.-H. Chen, "Disturbance rejection flight control for small fixed-wing unmanned aerial vehicles," *Journal of Guidance, Control and Dynamics*, vol. 39, no. 12, pp. 2804–2813, 2016.

[6] A. Anglade, J.-M. Kai, T. Hamel, and C. Samson, "Automatic control of convertible fixed-wing drones with vectorized thrust," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 5880–5887.

[7] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[8] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[9] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. Aparicio Ojea, E. Solowjow, and S. Levine, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5548–5555.

[10] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[11] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[12] E. Bøhn, E. M. Coates, S. Moe, and T. A. Johansen, "Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2019, pp. 523–533.

[13] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, 2019.

[14] X. Lin, Y. Yu, and C. Sun, "Supplementary reinforcement learning controller designed for quadrotor uavs," *IEEE Access*, vol. 7, pp. 26 422–26 431, 2019.

[15] Y. Wang, J. Sun, H. He, and C. Sun, "Deterministic policy gradient with integral compensator for robust quadrotor control," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 10, pp. 3713–3725, 2020.

[16] F. Fei, Z. Tu, D. Xu, and X. Deng, "Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyber-physical attacks," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7358–7364.

[17] N. Bernini, M. Bessa, R. Delmas, A. Gold, E. Goubault, R. Pennec, S. Putot, and F. Sillion, "A few lessons learned in reinforcement learning for quadcopter attitude control," in *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, 2021, pp. 1–11.

[18] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[19] J. Xu, T. Du, M. Foshey, B. Li, B. Zhu, A. Schulz, and W. Matusik, "Learning to fly: computational controller design for hybrid uavs with reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.

[20] W. Koch, R. Mancuso, and A. Bestavros, "Neuroflight: Next generation flight control firmware," *arXiv preprint arXiv:1901.06553*, 2019.

[21] R. J. Clarke, L. Fletcher, C. Greatwood, A. Waldock, and T. S. Richardson, "Closed-loop q-learning control of a small unmanned aircraft," in *AIAA Scitech 2020 Forum*, 2020, p. 1234.

[22] L. J. Fletcher, R. J. Clarke, T. S. Richardson, and M. Hansen, "Reinforcement learning for a perched landing in the presence of wind," in *AIAA Scitech 2021 Forum*, 2021, p. 1282.

[23] D. Wada, S. A. Araujo-Estrada, and S. Windsor, "Unmanned aerial vehicle pitch control under delay using deep reinforcement learning with continuous action in wind tunnel test," *Aerospace*, vol. 8, no. 9, 2021.

[24] S. Mysore, B. Mabsout, K. Saenko, and R. Mancuso, "How to train your quadrotor: A framework for consistently smooth and responsive flight control via reinforcement learning," *ACM Transactions on Cyber-Physical Systems (TCPS)*, vol. 5, no. 4, pp. 1–24, 2021.

[25] A. T. Azar, A. Koubaa, N. Ali Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, A. Ammar, B. Benjdira, A. M. Khamis, I. A. Hameed, and G. Casalino, "Drone deep reinforcement learning: A review," *Electronics*, vol. 10, no. 9, 2021.

[26] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft*. Princeton, NJ: Princeton University Press, 2012.

[27] E. Lavretsky and K. A. Wise, *Robust and Adaptive Control: With Aerospace Applications*. Springer London,

2013.

[28] S. Ferrari and R. F. Stengel, "Online adaptive critic flight control," *Journal of Guidance, Control, and Dynamics*, vol. 27, no. 5, pp. 777–786, sep 2004.

[29] Y. Zhou, E.-J. van Kampen, and Q. Chu, "Nonlinear adaptive flight control using incremental approximate dynamic programming and output feedback," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 493–496, feb 2017.

[30] C. Zhang, G. Zhang, and Q. Dong, "Adaptive dynamic programming-based adaptive-gain sliding mode tracking control for fixed-wing unmanned aerial vehicle with disturbances," *International Journal of Robust and Nonlinear Control*, oct 2022.

[31] D. Reinhardt, "On nonlinear and optimization-based control of fixed-wing unmanned aerial vehicles," Ph.D. dissertation, Norwegian University of Science and Technology, 2022.

[32] E. Bøhn, "Reinforcement learning for optimization of nonlinear and predictive control," Ph.D. dissertation, Norwegian University of Science and Technology, 2022.

[33] E. M. Coates, D. Reinhardt, K. Gryte, and T. A. Johansen, "Toward nonlinear flight control for fixed-wing uavs: System architecture, field experiments, and lessons learned," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2022, pp. 724–734.

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[35] E. Bøhn, "Project code and details," https://github.com/eivindeb/rluav, 2021.

[36] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[37] S. Mysore, B. Mabsout, R. Mancuso, and K. Saenko, "Regularizing action policies for smooth control with reinforcement learning," in *IEEE International Conference on Robotics and Automation*, 2021.

[38] K. Gryte, R. Hann, M. Alam, J. Roháč, T. A. Johansen, and T. I. Fossen, "Aerodynamic modeling of the sky-walker x8 fixed-wing unmanned aerial vehicle," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 826–835.

[39] E. M. Coates, A. Wenz, K. Gryte, and T. A. Johansen, "Propulsion system modeling for small fixed-wing uavs," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, pp. 748–757.

[40] A. Winter, R. Hann, A. Wenz, K. Gryte, and T. A. Johansen, "Stability of a flying wing uav in icing conditions," in *8th European Conference for Aeronautics and Aerospace Sciences (EUCASS)*, 2019.

[41] K. Gryte, "Precision control of fixed-wing uav and robust navigation in gnssdenied environments," Ph.D. dissertation, Norwegian University of Science and Technology,

2020.

[42] B. L. Stevens, F. L. Lewis, and E. N. Johnson, *Aircraft Control and Simulation*. Hoboken, NJ: Wiley-Blackwell, 2016.

[43] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 5055–5065.

[44] "ArduPilot open source drone software," https://ardupilot.org/, accessed: 2021-10-09.

[45] J. Pinto, P. S. Dias, R. Martins, J. Fortuna, E. Marques, and J. Sousa, "The lsts toolchain for networked vehicle systems," in *2013 MTS/IEEE OCEANS - Bergen*, 2013, pp. 1–9.

[46] A. Zolich, T. A. Johansen, K. Cisek, and K. Klausen, "Unmanned aerial system architecture for maritime missions. design amp; hardware description," in *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, 2015, pp. 342–350.

[47] A. B. Martinsen, "End-to-end training for path following and control of marine vehicles," Master's thesis, NTNU, 2018.

Appendix A
The ArduPlane Attitude Controller

This section presents the main equations used for attitude control in ArduPlane [44], which is a state-of-the-art open-source autopilot for fixed-wing UAVs. This controller is used as a baseline to compare the RL controller against and to support the discussion in Section V-C. The equations are based on ArduPlane, Release 4.0.9, which is the most recent stable release (as of August 2021).

The ArduPlane attitude controller consists of two cascaded single-input-single-output (SISO) feedback loops. The elevator controls pitch angle, while the ailerons are used for roll control. The outer loop consists of proportional controllers, where desired roll and pitch rates $p_r, q_r \in \mathbb{R}$ are calculated according to

$$p_r = k_\phi \left(\phi_r - \phi \right) \tag{22}$$
$$q_r = k_\theta \left(\theta_r - \theta \right) + q_{ct}, \tag{23}$$

where $k_\phi, k_\theta > 0$ and q_{ct} is the pitch rate offset needed to maintain a constant pitch angle during coordinated turns, given by

$$q_{ct} = \sin(\phi) \cos(\theta) \frac{g}{V_a} \tan(\phi). \tag{24}$$

The rate setpoints are inputs to the inner loop, which consists of proportional-integral (PI) controllers with feedforward

action:

$$\delta_a = k_{p,p}\nu^2 \left(p_r - p\right) + \int_0^t k_{i,p}\nu^2 \left(p_r - p\right) d\tau + k_{ff,p}\nu p_r \tag{25}$$

$$\delta_e = -k_{p,q}\nu^2 \left(q_r - q\right) - \int_0^t k_{i,q}\nu^2 \left(q_r - q\right) d\tau - k_{ff,q}\nu q_r, \tag{26}$$

where $k_{p,*}$, $k_{k_i,*}$ and $k_{ff,*}$ are proportional, integral and feedforward gains, respectively. The variable $\nu = V^*/V_a$, where V^* is some constant reference airspeed, provides airspeed scaling of the controller parameters, accounting for the fact that larger airspeeds give greater aerodynamic control authority. The negative sign in the control law for δ_e is introduced to account for the convention that positive elevator deflections yield a negative pitch moment [26].

For UAVs equipped with a rudder, additional control loops utilize the extra control surface for turn coordination. However, as the Skywalker X8 considered in this paper is rudderless, this part of the control algorithm is not relevant here.

For an elevon plane like the Skywalker X8, the aileron and elevator deflection angles are virtual control signals that are mapped to elevon control actions using the linear map

$$\delta_l = \delta_e + \delta_a \tag{27}$$
$$\delta_r = \delta_e - \delta_a. \tag{28}$$

By assuming a constant airspeed $V_a = 18\,\mathrm{m\,s^{-1}}$ and inserting parameters used for the Skywalker X8 UAV at the NTNU UAV-lab, we get the following sensitivities for the elevator and aileron control signals:

$$\left.\frac{\partial \delta_e}{\partial e_\theta}\right|_{\theta=\phi=0} = -1.0813 \qquad \frac{\partial \delta_a}{\partial e_\phi} = 1.6299 \tag{29}$$

$$\left.\frac{\partial \delta_e}{\partial q}\right|_{\theta=\phi=0} = 0.0312 \qquad \frac{\partial \delta_a}{\partial p} = -0.0243 \tag{30}$$

$$\left.\frac{\partial \delta_e}{\partial I_\theta}\right|_{\theta=\phi=0} = -0.0521 \qquad \frac{\partial \delta_a}{\partial I_\phi} = 0.0521 \tag{31}$$

$$\left.\frac{\partial \delta_e}{\partial \theta}\right|_{\theta=\phi=0} = 0.0104 \qquad \frac{\partial \delta_a}{\partial \phi} = -0.0104, \tag{32}$$

where $I_\phi = \int_0^t e_\phi d\tau$ and $I_\theta = \int_0^t e_\theta d\tau$ correspond to the (unbounded) integrator states of the RL controller.