Safe & Intelligent Control: Hybrid and Distributional Reinforcement Learning for Automatic Flight Control

Vieira dos Santos, L.; van Kampen, E.

DOI
10.2514/6.2025-2795

Publication date
2025

Document Version
Final published version

Published in
Proceedings of the AIAA SCITECH 2025 Forum

AIAA SciTech Forum
6-10 January 2025, Orlando, FL
AIAA SCITECH 2025 Forum

10.2514/6.2025-2795

Safe & Intelligent Control: Hybrid and Distributional Reinforcement Learning for Automatic Flight Control

Lucas Vieira dos Santos [1], Erik-Jan van Kampen[2]
Delft University of Technology, Kluyverweg 1 2629HS, Delft, The Netherlands

The critical challenge for employing autonomous control systems in aircraft is ensuring robustness and safety. This study introduces an intelligent and fault-tolerant controller that merges two Reinforcement Learning (RL) algorithms in a hybrid approach: the Distributional Soft Actor-Critic (DSAC) and the Incremental Dual Heuristic Programming (IDHP). The integration combines the strengths of DSAC in learning a robust control strategy and IDHP in allowing real-time control adaption. Compared to earlier controllers, such as a hybrid using the Soft Actor-Critic (SAC) algorithm and strictly offline DSAC and SAC, our hybrid demonstrates enhanced robustness against changing flight conditions and in the face of sensor noise and bias. During fault tolerance tests, it maintains superior control even when the effectiveness of the aircraft's ailerons and elevators is compromised. By demonstrating the potential of RL-based controllers to provide robustness and fault tolerance, this research advances the feasibility of safe and autonomous flight control operations.

I. Introduction

The advancements in Reinforcement Learning (RL) have led to remarkable achievements in various fields [1]. Notably, RL applications have outperformed humans in video games [2] and enabled robots to handle objects [3] and walk [4]. In the aerospace sector, RL can improve the robustness and fault-tolerance of autonomous systems. Those benefits are reflected in the growing research on RL-based control for aircraft [5], helicopters [6], and unmanned aerial vehicles [7] showing their potential in overcoming the limitations of traditional systems.

Though well-established, traditional Flight Control System (FCS) requires gain scheduling for different points of an aircraft's operational range, introducing complexity to the design and limiting adaptability to adverse conditions, such as system faults and malfunctions [8]. Moreover, Loss of Control (LOC) continues to be a major factor contributing to aviation accidents. Alone, LOC is responsible for 60 % of all fatalities in commercial aviation [9]. This underlines an urgent need for intelligent and adaptive control systems. RL-based FCS emerge as a promising solution. By learning from experience, these systems can offer better control autonomy, simplify the design process, and potentially reduce accidents caused by loss of control.

In the context of autonomous flight control, various RL-based controllers employing different algorithms and strategies have been explored. Offline algorithms, such as Soft Actor-Critic (SAC) and Distributional Soft Actor-Critic (DSAC), have demonstrated their capability to produce robust and efficient aircraft control strategies. However, these offline algorithms have their limitations, as they are not well-suited for adapting the strategies in-flight [10]. In contrast, online algorithms, such as IDHP, can adapt in real-time but raise questions about their robustness and the risks involved in learning while flying. Interestingly, a policy that combines SAC and IDHP has been shown by Teirlinck [11] to leverage the advantages of both algorithms.

[1]MSc Student, Faculty of Aerospace Engineering, Delft University of Technology. lucas6eng@gmail.com
[2]Associate Professor, Faculty of Aerospace Engineering, Delft University of Technology. E.vanKampen@TUDelft.nl

Most recently, DSAC has caught attention for its ability to create safer policies compared to SAC, as Seres [12] demonstrated the improvement of aircraft control in challenging conditions such as near stalls.

In light of these findings, this study develops an RL-based attitude control system that integrates the offline DSAC algorithm with the online IDHP algorithm. This hybrid model combines the robustness derived from pre-learned policies with the adaptability of online real-time learning. Differently than SAC, DSAC can learn complete distributions of possible outcomes providing deeper insights into the risks associated with different actions. During online operation, IDHP utilises the policy trained by DSAC and dynamically adjusts it according to the flight conditions.

The contributions of this paper are in developing safe and autonomous control systems by constructing fault-tolerant RL-based controllers. We assess the pros and cons of a hybrid offline-online controller, comparing DSAC and SAC as the offline component. Furthermore, we investigate how the hybrid controller contrasts with offline-only policies. At the core of this research is the innovative approach to flight control systems, which combines the hybrid and distributional reinforcement learning approaches to enhance safety without compromising performance.

The remainder of this article is organised as follows. Section II introduces the fundamental concepts of reinforcement learning and the algorithms employed to develop the hybrid controller. Section III delineates the formulation of the flight control task as a reinforcement learning problem and defines the hybrid controller. Section IV presents a discussion on the results of the experiments comparing the different controllers.

II. Background

This section establishes the Reinforcement Learning fundaments and notation, including the algorithms used for developing the hybrid attitude controller.

A. Fundamentals of Reinforcement Learning

Reinforcement Learning is a subfield of machine learning that primarily uses a trial-and-error approach to learn optimal strategies. In RL, an agent learns to make optimal decisions in a task by interacting with an environment. It chooses an action $\boldsymbol{a}_t \in \mathbb{R}^N$ at time t. This action changes the environment state from $\boldsymbol{s}_t$ to $s_{t+1} \in \mathbb{R}^M$ with a probability ρ. Then, the environment informs how satisfactory the state transition was by returning a scalar reward $r_{t+1} \in \mathbb{R}$ [13].

The objective for the agent is to identify the action it should select at each given time to maximise the total rewards, also known as return $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(\boldsymbol{s}_i, \boldsymbol{a}_i)$. The return includes a discount factor $\gamma \in [0, 1]$ that balances the objective of striving for near or long-term rewards.

Actor-critic is a specific type of RL algorithm that divides the tasks of policy learning and return estimation into two separate structures [14]. The Actor, the first structure, is a Neural Network (NN) that learns the policy π. It determines which action $\boldsymbol{a}_t$ should be taken given the state $\boldsymbol{s}_t$. For a stochastic policy, the action is sampled from $\boldsymbol{a}_t \sim \pi(\cdot \mid \boldsymbol{s}_t)$. In contrast, for a deterministic policy, the action is directly derived from $\boldsymbol{a}_t = \pi(\boldsymbol{s}_t)$.

The Critic, the second structure of actor-critic algorithms, approximates the expected return. There are two types of return functions the Critic can approximate: (1) the state-value function in Eq. (1), which informs the expected return for a given state, and (2) the action-value function in Eq. (2), which informs the expected return for a specific state-action pair:

$$V_\pi(\boldsymbol{s}_t) \doteq \mathbb{E}_\pi\left[R_t \mid \boldsymbol{s}_t\right] \tag{1}$$

$$Q_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) \doteq \mathbb{E}_\pi\left[R_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t\right] \tag{2}$$

Often, the action-value function is used in its recursive form, better known as the Bellman equation:

$$Q_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\rho, \boldsymbol{a}_{t+1}\sim\pi}\left[Q_\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})\right] \tag{3}$$

B. Incremental Dual Heuristic Programming (IDHP) Algorithm

Incremental Dual Heuristic Programming [15] is an online actor-critic RL algorithm characterised by three parametric structures: the Actor, the Critic and the Incremental Model. Differently from conventional actor-critic algorithms, IDHP incorporates an incremental model to learn an approximation of the system dynamics. The Incremental Model makes IDHP model-free, enabling the online learning process to use an approximation of the system dynamics. The Actor and Critic retain their traditional role in learning the policy and value function.

1. IDHP's Incremental Model

The incremental model learns a linear approximation of the system's non-linear state transition function, denoted as f. The first-order Taylor series expansion is used to approximate the state transition:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx \boldsymbol{x}_t + F_{t-1}\Delta\boldsymbol{x}_t + G_{t-1}\Delta\boldsymbol{u}_t \tag{4}$$

where, $\boldsymbol{x} \in \mathbb{R}^n$ represents the aircraft state vector, and $\boldsymbol{u} \in \mathbb{R}^m$ represents the action vector. F_{t-1} is the system transition matrix, and G_{t-1} is the control effectiveness matrix.

As IDHP is a model-free algorithm, the matrices F_{t-1} and G_{t-1} are unknown. The algorithm learns an approximation of those matrices with Recursive Least Squares (RLS). During each interaction with the environment, the incremental model uses the observed states to enhance the prediction of these matrices, thereby refining the estimation of the system's state transition.

To express the incremental model's update in matrix notation, we define the parameter matrix $\boldsymbol{\Theta} \in \mathbb{R}^{(n+m)\times n}$ as in Eq. (5), and the input information matrix $\boldsymbol{X}_t$ as in Eq. (6). With these matrices, the incremental model can predict the state transition with Eq. (7). The prediction error, denoted as $\boldsymbol{\epsilon}_t \in \mathbb{R}^{(1,n)}$, is computed with Eq. (8), which takes into account the observed change in state $\Delta\boldsymbol{x}_{t+1}$ from the environment.

$$\boldsymbol{\Theta}_{t-1} = \begin{bmatrix} F_{t-1}^T \\ G_{t-1}^T \end{bmatrix} \tag{5}$$

$$\boldsymbol{X}_t = \begin{bmatrix} \Delta\boldsymbol{x}_t \\ \Delta\boldsymbol{u}_t \end{bmatrix} \tag{6}$$

$$\Delta\hat{\boldsymbol{x}}_{t+1}^T = \boldsymbol{X}_t^T \cdot \hat{\boldsymbol{\Theta}}_{t-1} \tag{7}$$

$$\boldsymbol{\epsilon}_t = \Delta\boldsymbol{x}_{t+1}{}^T - \Delta\hat{\boldsymbol{x}}_{t+1}{}^T \tag{8}$$

The update to improve the Incremental Model's matrices requires computing the estimation covariance matrix Λ_t with Eq. (9). In this equation γ_{RLS} is the discount factor. It is important to note that this equation is recursive and relies on the covariance matrix from the previous timestep. Consequently, the covariance matrix is unknown at the start of the learning process and must be randomly initialised.

With the prediction error from Eq. (8) and the covariance error from Eq. (9), the Incremental Model's parameter matrix is updated according to Eq. (10). Subsequently, the model makes new predictions on the changes in the system's states and iteratively refines the matrices F and G until their convergence.

$$\Lambda_t = \frac{1}{\gamma_{\text{RLS}}}\left(\Lambda_{t-1} - \frac{\Lambda_{t-1}\boldsymbol{X}_t\boldsymbol{X}_t^T\Lambda_{t-1}}{\gamma_{\text{RLS}} + \boldsymbol{X}_t^T\Lambda_{t-1}\boldsymbol{X}_t}\right) \tag{9}$$

$$\hat{\boldsymbol{\Theta}}_t = \hat{\boldsymbol{\Theta}}_{t-1} + \frac{\Lambda_{t-1}\boldsymbol{X}_t}{\gamma_{\text{RLS}} + \boldsymbol{X}_t^T\Lambda_{t-1}\boldsymbol{X}_t}\boldsymbol{\epsilon}_t \tag{10}$$

2. IDHP's Actor-Network

The Actor-network of the IDHP agent is parameterised with weights denoted as θ. The architecture of this network includes a single hidden layer without a bias term. For attitude tracking, the reward is defined as the negative value of the tracking error, thereby penalising the agent for not following the reference signal. The Actor's loss, denoted as $\mathcal{L}_\pi$, is the negation value of the of the return:

$$\mathcal{L}_\pi(t) = -V(\boldsymbol{s}_t) = -[r_{t+1} + \gamma V(\boldsymbol{s}_{t+1})] \tag{11}$$

Updating the weights of the Actor-network, θ, requires computing the gradient of the loss function with respect to each layer in the network:

$$\nabla_\theta \mathcal{L}_\pi(t) = \frac{\partial \mathcal{L}_\pi(t)}{\partial \theta} = \frac{\partial \mathcal{L}_\pi(t)}{\partial \boldsymbol{a}_t}\frac{\partial \boldsymbol{a}_t}{\partial \theta} = -\left[\frac{\partial r_{t+1}}{\partial \boldsymbol{s}_{t+1}} + \gamma\hat{\lambda}(\boldsymbol{s}_{t+1})\right]\hat{G}_{t-1}\frac{\partial \boldsymbol{a}_t}{\partial \theta} \tag{12}$$

Here, $\hat{\lambda}(\boldsymbol{s}_{t+1})$ represents the Critic's prediction and $\partial \boldsymbol{a}_t/\partial\theta$ is computed using backpropagation through the Actor network. Subsequently, the weights of the Actor-Network are updated through gradient descent with a learning rate η_π.

3. IDHP's Critic-Network

The IDHP's Critic network λ is parametrised with weights denoted as ϕ. This network distinguishes itself from traditional critic networks by estimating the derivative of the value function with respect to each state of the aircraft, i.e., $\lambda = \partial V/\partial \boldsymbol{s}$, as opposed to estimating the value function directly. The loss of the Critic is calculated as the mean squared error of the Temporal Difference (TD):

$$\mathcal{L}_\lambda(t) = \frac{1}{2}\left[\lambda(\boldsymbol{s}_t) - \left(\gamma\hat{\lambda}(\boldsymbol{s}_{t+1}) + \frac{\partial r_{t+1}}{\partial \boldsymbol{s}_{t+1}}\right)\frac{\partial \boldsymbol{s}_{t+1}}{\partial \boldsymbol{s}_t}\right]^2 \tag{13}$$

The term $\partial \boldsymbol{s}_{t+1}/\partial \boldsymbol{s}_t$ is derived from the Incremental Model's approximation of the state transition dynamics.

The Critic's weights, denoted as ϕ, are updated by computing the gradient of the loss function with respect to the network layers:

$$\nabla_\phi \mathcal{L}_\lambda(t) = \frac{\partial \mathcal{L}_\lambda(t)}{\partial \phi} = \frac{\partial \mathcal{L}_\lambda(t)}{\lambda(\boldsymbol{s}_t)}\frac{\partial \lambda(\boldsymbol{s}_t)}{\partial \phi} = \left[\lambda(\boldsymbol{s}_t) - \left(\gamma\hat{\lambda}(\boldsymbol{s}_{t+1}) + \frac{\partial r_{t+1}}{\partial \boldsymbol{s}_{t+1}}\right)\left(F_{t-1} + G_{t-1}\frac{\partial \boldsymbol{a}_t}{\partial \boldsymbol{s}_t}\right)\right]\frac{\partial \lambda(\boldsymbol{s}_t)}{\partial \phi} \tag{14}$$

In this equation, $\partial \boldsymbol{a}_t/\partial \boldsymbol{s}_t$ is determined through backpropagation within the Actor-network. In contrast, $\partial\lambda(\boldsymbol{s}_t)/\partial\phi$ is computed by backpropagation through the Critic network. With the loss gradient, it is possible to update the weight values with gradient descent and a learning rate η_λ.

C. Soft Actor-Critic (SAC) Algorithm

The SAC [16, 17] algorithm is distinguished by three attributes. Firstly, it is an Actor-Critic RL algorithm, which implies that it learns an approximation for both the policy and the value function. Secondly, it is based on an off-policy formulation, which allows for the reuse of previously collected data, thereby enhancing sample efficiency. Lastly, it incorporates concepts from entropy maximisation into the RL objective, which promotes exploration and enhances stability.

1. SAC's Actor-Network

SAC introduces an entropy regularisation term to the conventional RL objective, resulting in the objective in Eq. (15). The additional entropy term $\mathcal{H}$ transforms the learning objective into a dual maximisation problem, emphasising both the expected return and the entropy of actions. As a result, this encourages the Actor to explore the environment more extensively.

$$\begin{aligned} \pi^* &= \underset{\pi}{\operatorname{argmax}} \sum_t \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi} [r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha \mathcal{H}(\pi(\cdot \mid \boldsymbol{s}_t))] \\ \text{with,} \quad & \mathcal{H}(\pi(\cdot \mid \boldsymbol{s}_t))] = \mathbb{E}_{\boldsymbol{a} \sim \pi(\cdot|\boldsymbol{s})} [-\log(\pi(\boldsymbol{a} \mid \boldsymbol{s}))] \end{aligned} \tag{15}$$

Here, the temperature parameter, denoted by α, balances the trade-off between entropy maximisation and expected return. Specifically, setting the temperature to zero reduces the objective to its conventional form, focusing exclusively on maximising expected return.

The actor in SAC aims to find the optimal policy defined in Eq. (15). The loss in achieving this objective is the negative of the value function. Therefore, it includes the Critic value estimation and the entropy term. This loss is shown in Eq. (16).

$$L_\pi = -V(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t \sim \pi} [\alpha \log \pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t) - Q(\boldsymbol{s}_t, \boldsymbol{a}_t)] \tag{16}$$

Nevertheless, the loss in Eq. (16) can lead to policies that make actions oscillate and change rapidly, especially in complex environments. To overcome this issue, we implement the Conditioning for Action Policy Smoothness (CAPS) [18] regularisation method, which adds two additional terms to the loss function, analogous to the approach adopted by Teirlinck [11].

The first term of the CAPS method is the temporal regularisation loss, which encourages each action to be near the immediate previous action. This temporal loss is defined in Eq. (17). The second term, called spatial regularisation, encourages the actions to be close to a randomly chosen action from the distribution $\bar{\boldsymbol{s}} \sim N(\boldsymbol{s}, 0.05)$. The spatial loss is defined in Eq. (18). It is important to note that temporal and spatial regularization distances are calculated using the L2 normalisation. By combining these CAPS terms with the actor loss, we get the full actor loss function shown in Eq. (19), where λ_T and λ_S are the scaling factors for the temporal and spatial terms, respectively.

$$\mathcal{L}_T = D(\pi(\boldsymbol{s}_t, \boldsymbol{s}_{t+1})) = \|\pi(\boldsymbol{s}_t) - \pi(\boldsymbol{s}_{t+1})\|_2 \tag{17}$$

$$\mathcal{L}_S = D(\pi(\boldsymbol{s}_t, \bar{\boldsymbol{s}})) = \|\pi(\boldsymbol{s}_t) - \pi(\bar{\boldsymbol{s}})\|_2 \tag{18}$$

$$\mathcal{L}_\pi^{\text{CAPS}} = \mathcal{L}_\pi + \lambda_T \mathcal{L}_T + \lambda_S \mathcal{L}_S \tag{19}$$

2. SAC's Critic-Network

The SAC's Critic estimates the action-value function Q with a NN parametrised with weights denoted by ϕ. The network aims to approximate the target value in Eq. (20). This target value is a function of the following state's reward and expected value, scaled by a discount factor γ. Even though the target value relies on the state-value function $V(\boldsymbol{s})$, there is no need to create a parameterisation for this function. This is because the state-value function and the action-value function are related, as demonstrated in Eq. (22). Therefore, the target value can be expressed in terms of the action-value function, as shown in Eq. (21).

$$y(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim p}[V(\boldsymbol{s}_{t+1})] \tag{20}$$

$$= r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \min_{i=1,2} [Q_{\phi_{\text{targ},i}}(\boldsymbol{s}_{t+1}, \tilde{\boldsymbol{a}}_{t+1}) - \alpha \log \pi_\theta(\tilde{\boldsymbol{a}}_{t+1} \mid \boldsymbol{s}_{t+1})] \tag{21}$$

$$V(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t \sim \pi}[Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \log \pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t)] \tag{22}$$

Note that Eq. (21) uses a Clipped Double Q-learning strategy to reduce overestimation bias. Therefore, it learns two separate Critic networks and takes the smaller value between them. Besides that, the equation uses Target Networks, which are updated less frequently than the actual Critic-Network, providing more stable learning. The $\tilde{\boldsymbol{a}}_{t+1}$ term indicates that the action is sampled from the actor instead of using an action from the replay buffer; therefore, $\tilde{\boldsymbol{a}}_{t+1} = \pi_\theta(\boldsymbol{s}_{t+1})$.

The Critic Network's learning process involves minimising its prediction's TD error. As such, the loss of the Critic Network, given in Eq. (23), is the squared difference between the estimated action-value function $Q(\boldsymbol{s}, \boldsymbol{a})$ and the target value y.

$$L_Q = \sum_{i=1,2} [Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - y(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})]^2 \tag{23}$$

D. Distributional Soft Actor-Critic (DSAC) Algorithm

The DSAC [19, 20] algorithm is an extension of the standard SAC.The key differentiation between DSAC and SAC lies in the learning approach adopted by the Critic Network: DSAC learns the distribution of returns rather than their expected values. Therefore, the Critic in DSAC approximates a function that describes the distribution of returns.

1. DSAC's Actor-Network

In DSAC, the Actor-Network denoted as π_θ, retains the same parameterised structure used in SAC. Therefore, the SAC's Actor loss function (as shown in Eq. (16))remains applicable to DSAC. However, since the Critic in DSAC estimates the distribution of returns $Z(\boldsymbol{s}, \boldsymbol{a})$, a transformation function is necessary to convert this distribution into a real-valued action-value function, Q. This transformation is facilitated through a risk measure function Ψ, represented as follows:

$$Q(\boldsymbol{s}, \boldsymbol{a}) = \Psi[Z(\boldsymbol{s}, \boldsymbol{a})] \tag{24}$$

One possible risk measure function could be an expectation operator, $\mathbb{E}[\cdot]$, which effectively converts the Actor loss into a conventional SAC format. Within the context of DSAC, the expectation risk measure is known as risk-neutral. This research uses the Wang risk measure [21]. Furthermore, the CAPS regularisation technique is added to the loss function the same way as done for the SAC

2. DSAC's Distributional Critic-Network

In the DSAC algorithm, the Critic Network aims to approximate the distribution of returns instead of the expectation of returns in the traditional SAC.Specifically, it approximates the quantile function $Z_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t; \tau_i)$, where τ_i denotes the i-th quantile, and ϕ represents the parameters of the NN. The quantile function, which is

the inverse of the Cumulative Distribution Function (CDF), indicates the value corresponding to a specific distribution quantile. For example, if $Z(10\%) = 5$, this indicates a 10% or lower probability of sampling a value of 5 or less from the given distribution.

To approximate the return distribution, the DSAC's Distributional Critic Network has multiple output neurons, each representing the quantile function at different quantiles. When combined, those quantile functions describe the distribution of return. The network input is the environment states, the Actor action, and the quantiles' values.

The error of the Critic Network is theTD error between two quantile fractions, as defined in Eq. (25). A quantile fraction, denoted as $\hat{\tau}$, is the mean value between two subsequent quantiles. Therefore, $\hat{\tau}_i = (\tau_i + \tau_{i+1})/2$. The TD error quantifies how the shape of the action-value distribution has changed after the action.

$$\delta_{ij}^t = r_t + \gamma \left[Z_{\bar{\phi}}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}; \hat{\tau}_i) - \log \pi_{\bar{\theta}}(\boldsymbol{a}_{t+1}, \boldsymbol{s}_{t+1}) \right] - Z_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t; \hat{\tau}_j) \tag{25}$$

The loss of the Critic Network in DSAC, denoted as $\mathcal{L}_Z$, is computed as the Huber loss $\mathcal{L}_k^H$ [22] of the TD error across all quantile pairs, as defined in Eq. (26). The Huber loss behaves as a mean squared error when the error value is smaller than the threshold k and behaves linearly beyond this threshold. This loss is chosen because it is less sensitive to outliers than pure mean squared error. The aggregate loss is calculated as a weighted sum of the Huber loss for each quantile pair, where the weight is the difference in the value of the quantile pair.

$$\mathcal{L}_Z = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (\tau_{i+1} - \tau_i) \mathcal{L}_k^H(\delta_{ij}^t) \tag{26}$$

III. Methodology

This section describes the methodology for developing and evaluating the RL-based attitude controllers.

A. Reinforcement Learning for Attitude Tracking Task

This study's Reinforcement Learning environment uses the DASMAT high-fidelity simulation of the Cessna 550 Citation II aircraft. The aircraft simulation has as input the states in Eq. (27) and as actions the vector in Eq. (28). Note that the aircraft states vector $\boldsymbol{x}$ is not the same as the environment observation vector $\boldsymbol{s}$, as the latter is tailored for each RL algorithm.

$$\boldsymbol{x} = \begin{bmatrix} p & q & r & V & \alpha & \beta & \theta & \phi & \psi & h \end{bmatrix}^T \tag{27}$$

$$\boldsymbol{u} = \begin{bmatrix} \delta_e & \delta_a & \delta_r \end{bmatrix}^T \tag{28}$$

In the state vector in Eq. (27), p is the roll rate, q the pitch rate, r the yaw rate, V the airspeed, α the angle of attack. β the sideslip angle, θ the pitch angle, ϕ the roll angle, ψ the heading angle, and h the altitude. In the action vector in Eq. (28), δ_e is the elvator deflection, δ_a the aileron deflection, and δ_r the rudder deflection.

The aircraft model and controller are discretised with a sampling frequency of 100 Hz. The model represents the aircraft in a clean configuration and is trimmed at 2000 m altitude and 90 m s^{-1} airspeed. Furthermore, the DASMAT model includes a built-in yaw damper and auto-throttle. Consequently, throttle control is managed by the model, while the RL agents manage the control surfaces in Eq. (28).

The main objective for the agents within the Citation environment is to control the aircraft such that its attitude match a reference signal. The reference attitude vector $\boldsymbol{x}^{\text{ref}}$, which represents the desired aircraft state, is

defined in Eq. (29). The actual aircraft attitude, $\boldsymbol{x}^{\text{att}}$ is extracted by filtering the state vector $\boldsymbol{x}$ as shown in Eq. (30).

$$\boldsymbol{x}^{\text{ref}} = \left[\theta^r, \phi^r, \beta^r\right]^T \tag{29}$$

$$\boldsymbol{x}^{\text{att}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x} \tag{30}$$

The tracking error is computed as the difference between the actual aircraft attitude $\boldsymbol{x}^{\text{att}}$ and the reference signal $\boldsymbol{x}^{\text{ref}}$. A weighted sum of the errors is employed to aggregate the tracking error across all attitude angles into a single metric, utilising the weights specified in Eq. (31). These weights serve the purpose of scaling each state, thereby ensuring they have comparable magnitudes. The values of the scaling factors employed in this study are retrieved from the work by Teirlinck [11]. Consequently, the attitude tracking error is defined in Eq. (32).

$$\boldsymbol{c}^{\text{att}} = \frac{180}{\pi}\left[\frac{1}{30} \quad \frac{1}{30} \quad \frac{1}{7.5}\right]^T \tag{31}$$

$$\boldsymbol{e}^{\text{att}} = (\boldsymbol{x}^{\text{att}} - \boldsymbol{x}^{\text{ref}}) \times \boldsymbol{c}^{\text{att}} \tag{32}$$

The offline (IDHP) and online (DSAC and SAC) agents are built with different reward functions and observation vectors. For the IDHP, the observation vector, defined in Equation Eq. (33), includes some of the aircraft states and the squared tracking error. The reward is calculated as the negative of the squared tracking error, as in Eq. (35). For the DSAC and SAC algorithms, use the observation vector in Eq. (34), and the reward function in Eq. (36).

$$\boldsymbol{s}_{t+1}^{\text{IDHP}} = \left[p \quad q \quad r \quad \alpha \quad \theta \quad \phi \quad \beta \quad \boldsymbol{e}_t^{\text{att}}\right]^T \tag{33}$$

$$\boldsymbol{s}_{t+1}^{\text{DSAC}} = \left[p \quad q \quad r \quad \boldsymbol{e}_t^{\text{att}}\right]^T \tag{34}$$

$$r_{t+1}^{\text{IDHP}} = -\boldsymbol{e}_t^{\text{att}} \times \boldsymbol{e}_t^{\text{att}} \tag{35}$$

$$r_{t+1}^{\text{DSAC}} = -\|\text{clip}[\boldsymbol{e}_t^{\text{att}}, -\vec{\mathbf{1}}, \vec{\mathbf{1}}]\| \tag{36}$$

B. Design of the Hybrid RL-Based Flight Controller

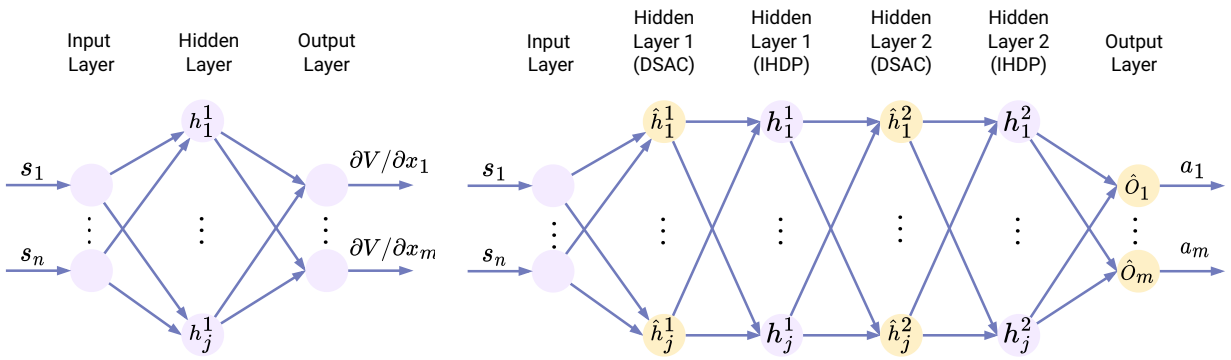
The controller developed in this study comprises a hybrid RL agent that combines the offline DSAC and online IDHP algorithms. This section describes the working principles underlying this controller.

1. Motivation For a Hybrid Controller

An autonomous flight controller must be robust and adaptable to handle unexpected conditions, such as system failures, during flight. Offline RL-based controllers are robust but inefficient in adapting to changing conditions. Online RL controllers are good at adapting in real-time but can be unreliable, and there are safety concerns as they learn during the flight. This study introduces a Hybrid controller that uses IDHP and DSAC to bring together their strengths.

The research by Teirlinck [11] demonstrated the advantages of a hybrid controller that combines IDHP with SAC. Recent studies [12] suggest that DSAC is a safer option than SAC without sacrificing performance. DSAC, being risk-sensitive, tends to avoid states with high uncertainties, as Seres [12] demonstrated a DSAC controller that avoided near-stall conditions.

DSAC also exhibits more stable learning performance. It is more consistent and reliable in learning an optimal policy with a high return and low variance. This research aims to evaluate the performance of a hybrid controller that uses DSAC as an offline algorithm, comparing it with standalone DSAC and SAC, as well as the hybrid that uses SAC.



(a) Hybrid's Critic-Network. **(b) Hybrid's Actor-Network.**

Figure 1. Topology of DSAC-hybrid's Critic and Actor-Network. $\hat{h}$ are the hidden layers derived from the DSAC agent, which are frozen during online learning. h are the layers from the IDHP agent, which can adapt during online learning.

2. Naming Conventions for Hybrid Controllers

Throughout this paper, the SAC algorithm combined with IDHP is referred to as SAC-hybrid, while its standalone version is called SAC-only. Likewise, when DSAC is combined with IDHP, it is called IDHP-hybrid, and DSAC-only when it operates alone.

3. Hybrid Controller architecture

The hybrid controller's architecture is based on the one developed by Teirlinck [11]. In this setup, the Actor-Network of the IDHP algorithm is modified to include neurons from the offline algorithm. That means layers from both offline agent and IDHP are intertwined in a single network. The layers derived from the offline algorithm retain the weight values learned during offline training, while IDHP layers are set to the identity matrix at the beginning. This ensures that at time zero, the Actor-network functions as the network of the offline agent.

The Hybrid Actor Network, which combines layers from IDHP and DSAC, is shown in Fig. 1b. During the online learning phase, the network only updates the IDHP layers, while the layers associated with the offline algorithm remain unchanged. Because only the layers derived from IDHP change, the algorithm update logic discussed in Section II.B does not alter.

On the other hand, the Critic Network of the Hybrid controller does not combine layers from the offline and online algorithms. This network is the same as the one from IDHP, as shown in Fig. 1a. This design choice is due to the IDHP's Critic output, which is significantly different from the one in SAC and DSAC. Consequently, reusing the offline Critic would change the logic of the IDHP algorithm.

C. Training Process for the Hybrid Controller

To create the Hybrid agents, the SAC or DSAC algorithms are first trained offline. For this study, both algorithms were trained over three random seeds with f 1 M steps each. The training episode consisted of 2 K steps(or 20 s) where the agents should track a smoothed step function with variable amplitude. Figure 2 shows the averaged learning curves of the two algorithms across the three random seeds. All agents converged to similar performance levels, with SAC achieving convergence slightly sooner than DSAC.

During offline training, the agents update their networks to improve tracking performance. After each episode,

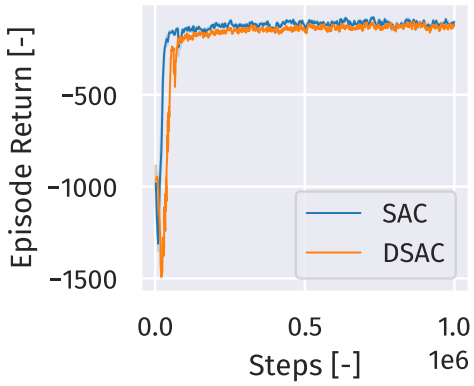


Figure 2. Learning performance curves for the SAC and DSAC algorithms. Each curve represents the average performance of three independent runs. The training was conducted over a total of 10 M **learning steps, partitioned into episodes of** 2 K **steps each.**

the policy is evaluated. The optimal policy at the end of training is the one that showed the best performance during evaluation. The metric used for the evaluation is the normalised Mean Absolute Error (nMAE). This metric normalises the tracking error by dividing it by the range of each tracked state, which enables the comparison of scores across different tracking tasks. The policies learned during this process are shown in Table 1 alongside their best nMAE. It is important to note that, the nMAE quantifies the normalised error; therefore, a lower value indicates superior performance.

Table 1. Evaluation performance of SAC and DSAC agents during training. This table provides the best nMAE score the agents achieved during evaluation after each episode in the training process. The training was conducted over a total of 10 M **learning steps, partitioned into episodes of** 2 K **steps each.**

Id	Agent	nMAE	Id	Agent	nMAE
SAC-1	SAC	8.36 %	DSAC-1	DSAC	7.57 %
SAC-2	SAC	10.02 %	DSAC-2	DSAC	11.80 %
SAC-3	SAC	8.03 %	DSAC-3	DSAC	7.03 %

IV. Results & Discussion

In this section, we present the results of evaluating the performance of the proposed hybrid controller. These results clarify the controller's robustness to changes in reference signals and under conditions of sensor noise and bias, as well as its fault tolerance when subjected to reductions in control surface effectiveness.

A. Analysis of Robustness

This subsection is dedicated to examining the robustness of the hybrid controller under different reference signals and in the presence of sensor noise and bias.

1. Robustness to Different Reference Signals

An important goal for the RL-based controller is to ensure that the policy it learns is general enough to track reference signals different from those used during training. To better understand how the hybrid and

non-hybrid controllers behave under different conditions, we compare their performance in response to three distinct reference signals, shown in Fig. 3.

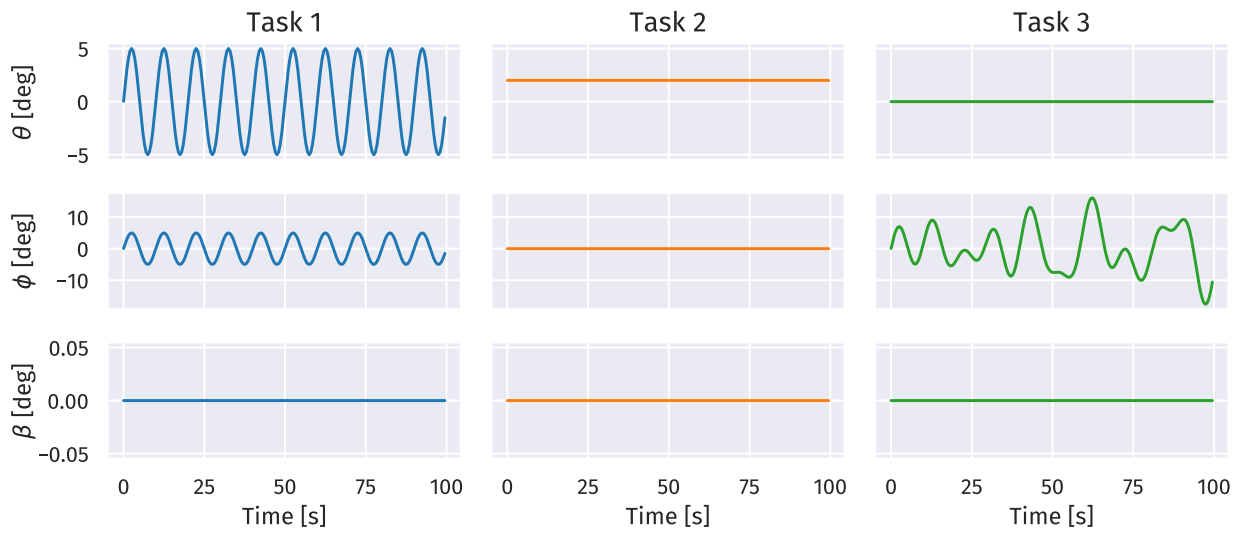


Figure 3. The three tracking tasks used in the robustness experiment, each providing a different attitude reference.

The first reference signal requires tracking a sinusoidal signal with a fixed amplitude and period for the pitch (θ) and roll (ϕ) angles while maintaining the sideslip angle (β) at zero. The second task involves keeping the three attitude angles constant throughout the flight. The third task involves holding both the pitch and sideslip angle at zero while the roll angle follows a pseudo-random sinusoidal trajectory.

The robustness of the agents across different tracking tasks is summarised in Table 2. Each agent was evaluated in a task for five different random seeds using each of the three offline-learned policies. Therefore, a total of 15 runs for each agent. The results indicate that, on average, both SAC-hybrid and DSAC-hybrid improved the performance of their respective offline-only versions.

Furthermore, the table also shows how the nature of the tracking task can influence the algorithms' performance. When the evaluation task is significantly different from the trained task, the agent may not be able to achieve low tracking errors. This is particularly evident in Task 1, where the hybrid agents improve performance compared to the offline agents but are less substantial than in the other tasks.

Table 2. Evaluation of hybrid and non-hybrid agents subjected to varying tracking tasks. The table presents the mean and variance of their tracking nMAE.

	SAC-only	**SAC-hybrid**	**DSAC-only**	**DSAC-hybrid**
Task 1	20.1 ± 1.8%	14.9 ± 1.5%	22.5 ± 7.0%	11.3 ± 0.6%
Task 2	11.1 ± 3.6%	4.5 ± 1.8%	13.4 ± 8.8%	2.9 ± 0.9%
Task 3	16.6 ± 2.2%	4.1 ± 0.8%	18.7 ± 9.8%	3.0 ± 0.7%

It is noteworthy from the results that, between the offline agents, the SAC-only outperforms the DSAC-only in all tasks, with lower variance. However, in hybrid form, the DSAC-hybrid manages to surpass the SAC-hybrid performance. The hybrid configuration enhances the offline policies, making them more robust. The key

takeaway is that the hybrid setup yields more consistent tracking performance results, irrespective of the initial performance of the offline model.

Across all the agents, the DSAC-hybrid consistently achieved the lowest average nMAE in all tasks while also maintaining low variance. Figure 4 shows the extent of improvement in nMAE performance that the hybrid agent brings compared to its offline-only version. The figure reveals that, on average, the DSAC-hybrid enhances performance more effectively than the SAC-hybrid. However, this improvement comes with higher variance. Additionally, during Task 2, the DSAC-hybrid had one run using the DSAC-2 offline policy (as referenced in Table 1), which resulted in reduced performance. This occurred because the DSAC-2 offline policy had the lowest nMAE, highlighting the importance of carefully selecting the initial policy for the hybrid, favouring those with low nMAE.

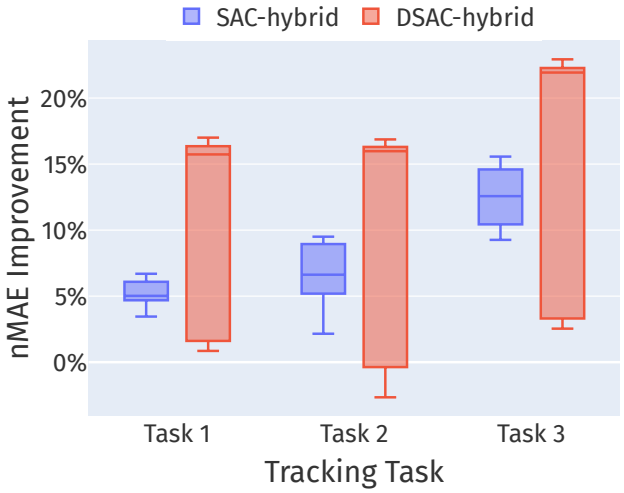


Figure 4. Box plot showing the extent of improvement innMAE that each hybrid algorithm (SAC-hybrid and DSAC-hybrid) brings compared to their respective offline algorithms across different tasks during the evaluation phase.

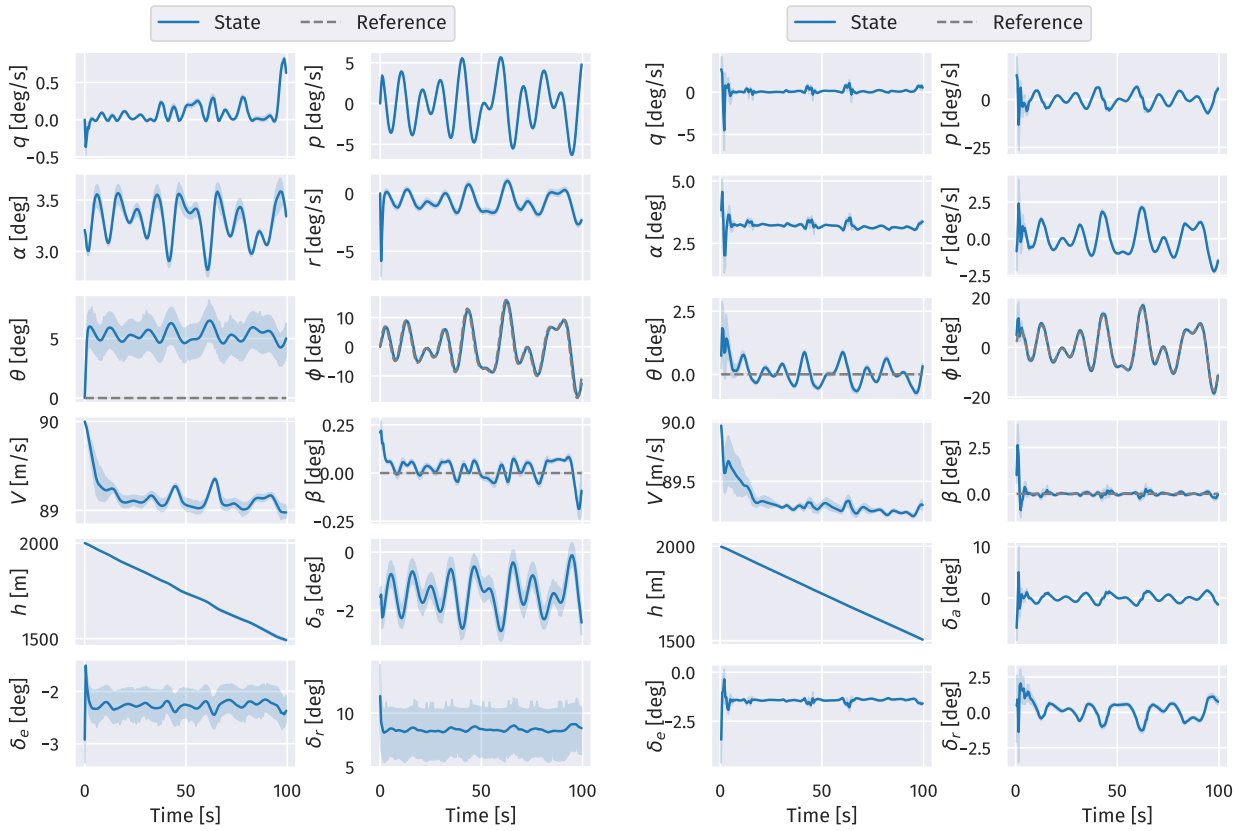
2. Robustness under Sensor Noise and Bias

The final robustness experiment involves testing the agents in a more realistic setting by introducing noise and bias into the aircraft's sensors. In this test, the hybrid agents aim to track a reference signal, while noise and bias are added to the states p, q, and r. The noise and bias are sampled from the normal distribution $\mathcal{N}(\mu = 3 \cdot 10^{-5}, \sigma = 4 \cdot 10^{-7})$, which are based on expected values for the Citation aircraft as found in Grondman *et al.* [23]. Only these three states are modified because they are the only ones shared between the observation functions of the offline algorithms and IDHP, as shown in Eq. (34) and Eq. (33), respectively.

Each offline learned policy was evaluated in a hybrid configuration under noise and bias conditions using five different random seeds for this experiment. This amounts to 15 runs for SAC and 15 runs for DSAC. The results from this experiment are summarised in Table 3. Despite the presence of noise and bias, both hybrid controllers managed to enhance the performance compared to the offline-only versions. The DSAC algorithm again showed a more substantial improvement. The graphs showing the average tracking performance of the agents during the task with noise and bias can be seen in Fig. 5a for DSAC-only, and in Fig. 5b for DSAC-hybrid.

Table 3. Evaluation of hybrid and non-hybrid agents subjected to sensor noise and bias. The table presents the mean and variance of their tracking nMAE.

	Non-hybrid	Hybrid	Improvement
SAC	16.6 ± 2.2%	4.0 ± 0.9%	12.6 ± 2.7%
DSAC	18.7 ± 9.8%	2.8 ± 0.5%	15.9 ± 9.5%



(a) DSAC-only evaluation performance. **(b) DSAC-hybrid evaluation performance.**

Figure 5. Aircraft states during the evaluation of DSAC agents in Task 3, including sensor noise and bias.

B. Assessment of Fault Tolerance

The fault-tolerance experiment examines the performance of the hybrid controllers under reduction in the effectiveness of its control surfaces. First we consider a the elevator effectiveness by 70 %. Then, a reduction in the aileron effectiveness by 90 %. In each condition, the agents are evaluated over 5 random seeds, each tracking Task 3 from Fig. 3 while the fault stats at t =10 s.

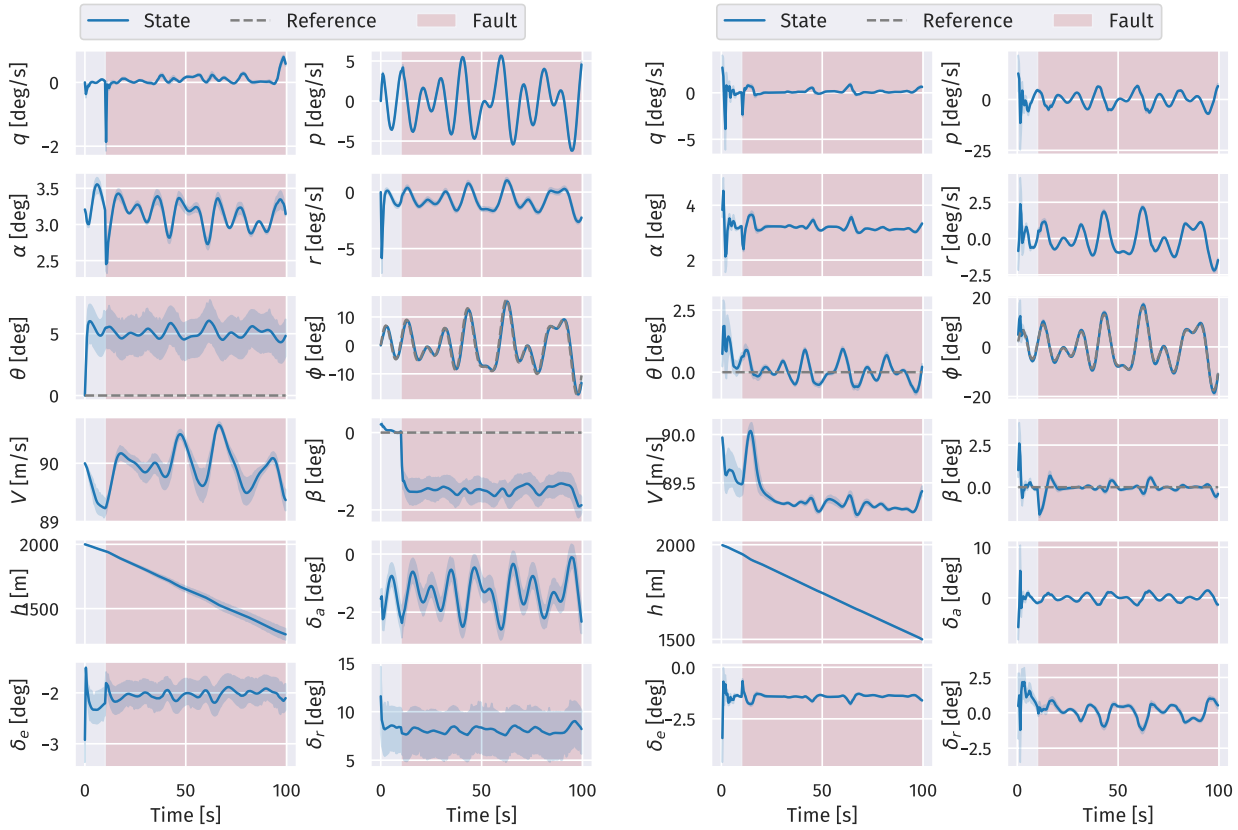
1. Performance under Reduced Elevator Effectiveness

The evaluation of the hybrid and non-hybrid agents with reduction in the elevator effectiveness by 70 % at time 10 s is summarised in Table 4. Those results shows the hybrid compensate to the failure and improve the performance of the offline-only agents. The DSAC-hybrid achieves the lowest nMAE.

Table 4. Evaluation of hybrid and non-hybrid agents subjected to a reduction of elevator effectivess by 70 %. The table presents the mean and variance of their tracking nMAE.

	Non-hybrid	Hybrid	Improvement
SAC	18.4 ± 3.1%	4.1 ± 0.9%	14.3 ± 3.7%
DSAC	22.4 ± 11.8%	3.2 ± 0.6%	19.2 ± 11.4%

The average tracking performance of the DSAC-hybrid and DSAC-only agents is shown in Fig. 6. While DSAC-only does adequate tracking of the roll angle, it struggles to hold the pitch angle and sideslip angle at zero. For the sideslip angle it is visible that the deterioration start at the start of the fault. The DSAC-hybrid, on the other hand, has a worst start on the tracking of the roll angle, but it picks up the tracking of the signal within the first seconds. Additionaly, it is also visible the impact of the fault on pitch and sideslip angle at start of fault. However, the hybrid is able to account for the fault and do a better regulate those angles.



(a) DSAC-only evaluation performance. **(b) DSAC-hybrid evaluation performance.**

Figure 6. Aircraft states during the evaluation of DSAC agents in Task 3, including a reduction in elevator effectiveness by 70 % at the 10 s mark.

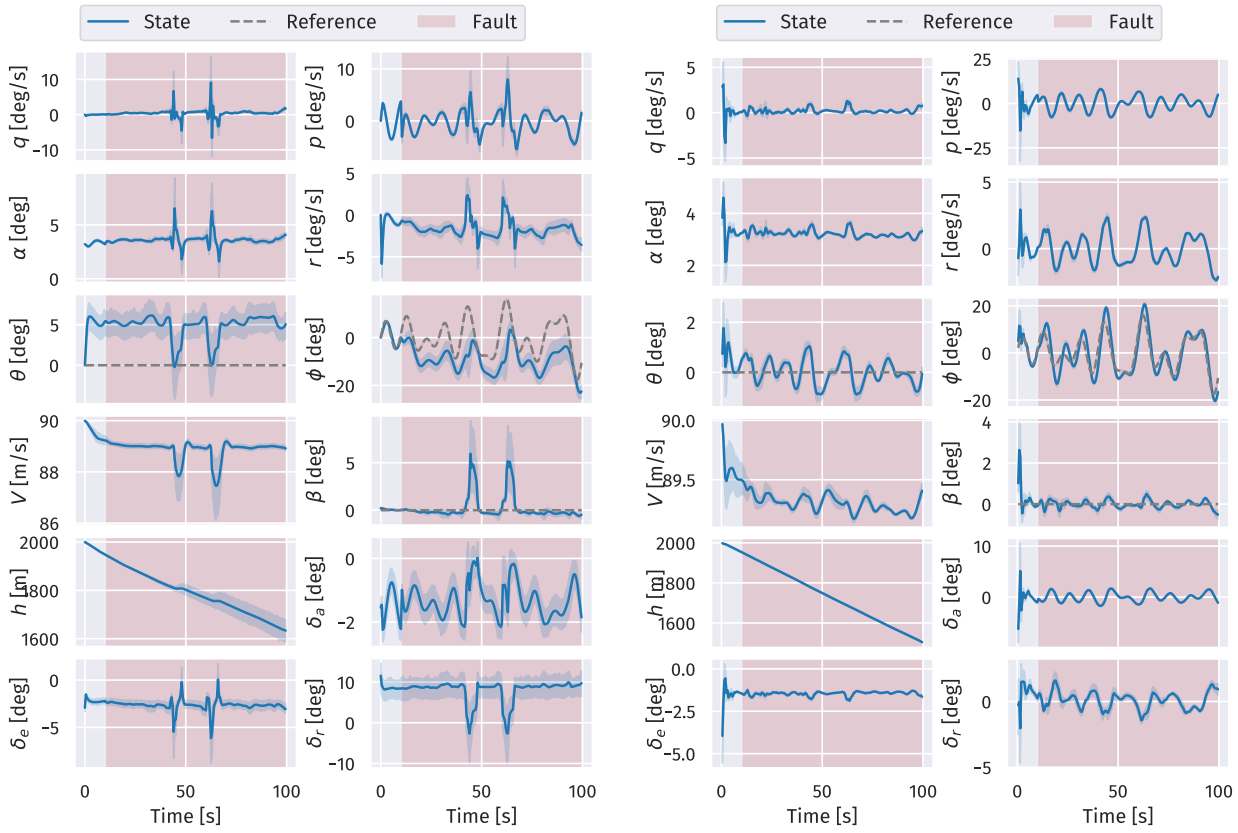
2. Performance under Reduced Aileron Effectiveness

The evaluation of the hybrid and non-hybrid agents with a reduction in the aileron effectiveness by 90 % at time 10 s is summarised in Table 5. The results show that DSAC-hybrid is the controller that achieves the lowest nMAE, followed by SAC-hybrid. This is the experiment where the hybrid architecture showed the greatest nMAE improvement concerning the non-hybrid agents.

Table 5. Evaluation of hybrid and non-hybrid agents subjected to a reduction of aileron effectiveness by 90 %. The table presents the mean and variance of their tracking nMAE.

	Non-hybrid	Hybrid	Improvement
SAC	28.5 ± 8.5%	9.1 ± 2.5%	19.4 ± 10.7%
DSAC	28.7 ± 17.0%	6.1 ± 0.7%	22.6 ± 16.8%

The results of the average tracking performance of the DSAC-hybrid in the evaluation task with the aileron fault is shown in Fig. 7. With the deteriorated aileron, the DSAC-only strugles to track the roll angle, as shown in Fig. 7a. Note how different the tracking becomes when compared to Fig. 7b. The DSAC-hybrid, however, is able to track the roll angle considerably well while still managing to hold the other attitude angles.



(a) DSAC-only evaluation performance.

(b) DSAC-hybrid evaluation performance.

Figure 7. Aircraft states during the evaluation of DSAC agents in Task 3, including a reduction in aileron effectiveness by 90 % at the 10 s mark.

C. Reliability Analysis of the Controllers

One of the challenges in RL is evaluating the reliability of results. Because the training time is computationally intensive, the number of experiments that can be practically conducted is limited. Consequently, it becomes difficult to assert that the results are reliable and not only an outlier due to randomness in the problem. To address this issue, the study performed a reliability analysis to evaluate the likelihood of obtaining similar results if the same experiments were performed with different random seeds.

The methodology employed for this reliability analysis uses the framework set by Agarwal *et al.* [24]. A central component in the analysis is the stratified bootstrap confidence intervals, which is an aggregated score built by random sampling, with replacement, a set of nMAE values from each experiment. The sample size is proportional to the number of evaluations in that experiment.

1. Performance Profile

The performance profile graph shows the distribution of nMAE across all runs. It uses the data derived from the stratified bootstrap confidence intervals to build a cumulative distribution of the model's scores. The results for the agents are shown in Fig. 8a. The y values in the curve represent the fraction of runs that lead to an nMAE lower than the threshold in the x axis. Therefore, the higher the curve, the better.

In the performance profile graph, the hybrid agents exhibit comparable performance relative to one another, as do the non-hybrid agents. The curve for the DSAC-hybrid is above the others, signifying that this agent demonstrates statistical dominance. This implies that, for this particular agent, the runs are more likely to yield lower nMAE. Among the non-hybrid algorithms, SAC-only exhibits slightly better performance compared to DSAC-only.

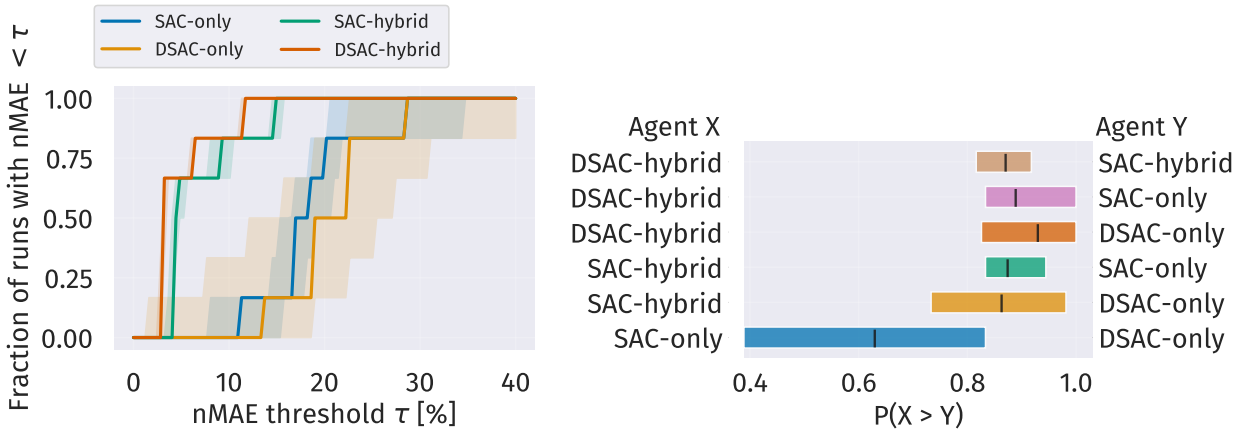
2. Probability of Improvement

A subsequent analysis is the probability of improvement, which quantifies the likelihood that a given algorithm will yield performance improvement compared to others. In other words, it calculates the probability that the performance of algorithm X exceeds that of algorithm Y. The calculation involves a pairwise comparison of the nMAE between the two algorithms, isolating instances where the performance of X surpasses that of Y. The probability of improvement is then computed as the ratio of instances where X performed better to the total number of pairwise comparisons made.

The outcomes of the probability of improvement analysis for the algorithms are shown in Fig. 8b. The data indicate that both hybrid algorithms have greater than 80 % probability of increasing the performance of the offline-only agents. Furthermore, the DSAC-hybrid also has a high probability of improving the SAC-hybrid. Among the offline-only agents, SAC has a slightly above 60 % chance of improving DSAC. However, this is together with high variance, confirming that the performance of those offline algorithms is similar.

V. Conclusion

Robustness and fault tolerance are critical for autonomous flight control systems. Reinforcement Learning (RL) flight controllers emerge as a promising method to achieve these characteristics. While offline RL algorithms are good in providing robustness, they are constrained by a limited capacity to adapt during flight. In contrast, online RL algorithms are highly adaptive but may compromise reliability due to uncertain convergence properties and the inherent risks associated with learning to control during flight. In light of these observations, this research investigated the potential of hybrid controllers, combining the strengths of offline Distributional Soft Actor-Critic (DSAC) and online Incremental Dual Heuristic Programming (IDHP) algorithms, referred to as DSAC-hybrid.



(a) Performance profile of RL agents. **(b) Probability of improvement of RL agents.**

Figure 8. Reliability analysis of the hybrid and non-hybrid controllers. Results from studying the stratified bootstrap confidence intervals of the aggregated nMAE scores across the experiments.

The DSAC-hybrid controller demonstrated enhanced robustness compared to a hybrid using Soft Actor-Critic (SAC) as the offline algorithm (denoted to as SAC-hybrid). Besides that, the DSAC-hybrid outperformed controllers exclusively utilising DSAC and SAC. The robustness comparison included evaluating the controllers' ability to track reference signals that deviate from those encountered during training and in scenarios with noisy and biased observation vectors.

Furthermore, the DSAC-hybrid controller showed superior fault-tolerance capabilities. Specifically, in an assessment involving a scenario where elevator effectiveness was reduced by 70 %, the DSAC-hybrid maintained its capacity to track a reference signal. At the same time, the strictly offline agents encountered difficulties. The same was observed with a 90 % reduction in aileron effectiveness. The SAC-hybrid agent also coped with the fault effectively but with lower performance than the DSAC-hybrid.

Overall, the hybrid approach proved to effectively combine offline algorithms' robustness with an online algorithm's real-time adaptability. The DSAC-hybrid emerged as the highest performance model, with SAC-hybrid also demonstrating superiority over the strictly offline algorithms. A reliability analysis provided additional validation to these conclusions, indicating that both hybrid algorithms have a greater than 80 % likelihood of outperforming offline algorithms regarding tracking performance.

This research is a step towards developing safe and autonomous RL-based controllers. By combining offline and online learning algorithms, the hybrid approach opens up new possibilities for flight control. In particular, the DSAC-hybrid controller sets an example of robustness and adaptiveness in attitude tracking. Therefore, these findings have significant implications for future research in autonomous flight control.

References

[1] M. Naeem, S. T. H. Rizvi, and A. Coronato, "A Gentle Introduction to Reinforcement Learning and its Application in Different Fields," *IEEE Access*, vol. 8, pp. 209 320–209 344, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3038605.

[2] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 26, 2015, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14236.

[3] OpenAI *et al.* "Learning Dexterous In-Hand Manipulation." arXiv: 1808.00177 [cs, stat]. (Jan. 18, 2019), [Online]. Available: http://arxiv.org/abs/1808.00177 (visited on 04/12/2023), preprint.

[4] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. "Learning to Walk via Deep Reinforcement Learning." arXiv: 1812.11103 [cs, stat]. (Jun. 19, 2019), [Online]. Available: http://arxiv.org/abs/1812.11103 (visited on 04/12/2023), preprint.

[5] J. H. Lee and E.-J. Van Kampen, "Online reinforcement learning for fixed-wing aircraft longitudinal control," in *AIAA Scitech 2021 Forum*, VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, Jan. 11, 2021, ISBN: 978-1-62410-609-5. DOI: 10.2514/6.2021-0392.

[6] A. Y. Ng *et al.*, "Autonomous Inverted Helicopter Flight via Reinforcement Learning," in *Experimental Robotics IX*, M. H. Ang and O. Khatib, Eds., red. by B. Siciliano, O. Khatib, and F. Groen, vol. 21, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 363–372, ISBN: 978-3-540-28816-9 978-3-540-33014-1. DOI: 10.1007/11552246_35.

[7] 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, Apr. 30, 2019, ISSN: 2378-962X, 2378-9638. DOI: 10.1145/3301273.

[8] P. C. Gregory, "Proceedings of the self adaptive flight control systems symposium," Aeronautical Systems Div Wright-Patterson AFB OH Flight Control Lab, United States, Technical Rerport AD0209389, 1959.

[9] IATA, *Loss of Control In-Flight Accident Analysis Report*. Montreal: International Air Transport Association, 2019, ISBN: 978-92-9264-002-6.

[10] K. Dally and E.-J. Van Kampen, "Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control," in *AIAA SCITECH 2022 Forum*, San Diego, CA & Virtual: American Institute of Aeronautics and Astronautics, Jan. 3, 2022, ISBN: 978-1-62410-631-6. DOI: 10.2514/6.2022-2078.

[11] C. Teirlinck, "Reinforcement Learning for Flight Control Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance," TU Delft, Delft, 2022.

[12] P. Seres, "Distributional Reinforcement Learning for Flight Control," Delft University of Technology, Delft, 2022.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018, ISBN: 0-262-03924-9.

[14] P. J. Werbos, "A menu of designs for reinforcement learning over time," in *Neural Networks for Control*, ser. Neural Network Modeling and Connectionism, W. T. Miller, R. S. Sutton, P. J. Werbos, and N. S. F. (U.S.), Eds., Cambridge, Mass: MIT Press, 1990, pp. 67–97, ISBN: 978-0-262-13261-9.

[15] Y. Zhou, E.-J. van Kampen, and Q. P. Chu, "Incremental model based online dual heuristic programming for nonlinear adaptive control," *Control Engineering Practice*, vol. 73, pp. 13–25, Apr. 2018, ISSN: 09670661. DOI: 10.1016/j.conengprac.2017.12.011.

[16] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," version 2, 2018. DOI: 10.48550/ARXIV.1801.01290.

[17] T. Haarnoja *et al.*, "Soft Actor-Critic Algorithms and Applications," version 2, 2018. DOI: 10.48550/ARXIV.1812.05905.

[18] S. Mysore, B. Mabsout, R. Mancuso, and K. Saenko, "Regularizing Action Policies for Smooth Control with Reinforcement Learning," version 2, 2020. DOI: 10.48550/ARXIV.2012.06644.

[19] X. Ma, L. Xia, Z. Zhou, J. Yang, and Q. Zhao. "DSAC: Distributional Soft Actor Critic for Risk-Sensitive Reinforcement Learning." arXiv: 2004.14547 [cs]. (Jun. 10, 2020), (visited on 10/21/2022), preprint.

[20] J. Duan, Y. Guan, S. E. Li, Y. Ren, and B. Cheng, "Distributional Soft Actor-Critic: Off-Policy Reinforcement Learning for Addressing Value Estimation Errors," version 3, 2020. DOI: 10.48550/ARXIV.2001.02811.

[21] S. S. Wang, "A Class of Distortion Operators for Pricing Financial and Insurance Risks," *The Journal of Risk and Insurance*, vol. 67, no. 1, p. 15, Mar. 2000, ISSN: 00224367. DOI: 10.2307/253675. JSTOR: 253675.

[22] P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, Mar. 1964, ISSN: 0003-4851. DOI: 10.1214/aoms/1177703732.

[23] F. Grondman, G. Looye, R. O. Kuchar, Q. P. Chu, and E.-J. Van Kampen, "Design and Flight Testing of Incremental Nonlinear Dynamic Inversion-based Control Laws for a Passenger Aircraft," in *2018 AIAA Guidance, Navigation, and Control Conference*, Kissimmee, Florida: American Institute of Aeronautics and Astronautics, Jan. 8, 2018, ISBN: 978-1-62410-526-5. DOI: 10.2514/6.2018-0385.

[24] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare. "Deep Reinforcement Learning at the Edge of the Statistical Precipice." arXiv: 2108.13264. (Jan. 5, 2022), [Online]. Available: http://arxiv.org/abs/2108.13264 (visited on 04/28/2023), preprint.

Appendix

A. DSAC and IDHP Hyperparameters

This section provides the hyperparameters employed during the training process of the Reinforcement Learning algorithms used in this research. The DSAC hyperparameters are shown in Table 6a and IDHP ones in Table 6b.

(a) Hyperparameters for DSAC algorithm, adapted from Seres [12] and Duan *et al.* [20]

Hyperparameter	Symbol	Value
Learning rate	η	$4.4 \cdot 10^{-4}$
Discount Factor	γ	0.99
Hidden Neurons		64×64
Replay buffer size	$\lvert\mathcal{D}\rvert$	$1 \cdot 10^{6}$
Batch Size	$\lvert\mathcal{B}\rvert$	256
Polyak Step		0.995
Optimiser		Adam
Network Activation		ReLu
CAPS Scaling	λ_T, λ_S	400
Nr. of Quantiles		8
Huber Threshold	k	1
Risk Measure	Ψ	Wang

(b) Hyperparameters for IHDP algorithm, adapted from Teirlinck [11] and Zhou *et al.* [15]

Hyperparameter	Symbol	Value
Actor Learning Rate	η_A	0.1
Critic Learning Rate	η_C	0.001
Discount Factor	γ	0.7
RLS Discount Factor	γ_{RLS}	0.7
Hidden Neurons		10
Optimiser		SGD
Network Activation		Tanh

Table 6. Value of hyperparameter for DSAC and IDHP algorithms.