End-to-End Hierarchical Reinforcement Learning for Adaptive Flight Control

A method for model-independent control through Proximal Policy Optimization with learned Options

Zhouxin Ge

Technische Universiteit Delft

TU Delft

End-to-End Hierarchical Reinforcement Learning for Adaptive Flight Control

A method for model-independent control through Proximal Policy Optimization with learned Options

by

Zhou Xin Ge

to obtain the degree of Master of Science in Aerospace Engineering
at the Delft University of Technology,
to be defended publicly on Friday August 27, 2021 at 09:30 AM.

Student number:	4272358	
Thesis committee:	Prof. dr. G.C.H.E. de Croon	TU Delft, Chairman
	Dr. ir. E. van Kampen,	TU Delft, Supervisor
	Dr. ir. M.A. Mitici,	TU Delft, External Examiner

Cover picture depicting an paper airplane being an abstraction of a realistic airplane - credits: inspired by [69]

TU Delft

Preface

It's a wrap! Doing a thesis during the pandemic was a challenge in itself and required some 'adaptivity' to be familiar with a new way of working and sharing ideas. This journey has taught me a lot and I could not have done it without the help of the great people around me.

Erik-Jan, thank you for your support and guidance throughout this project and making time for me in your holidays. Bart, Marijn and Max thank you for the always fun and 'gezellige' moments during breakfast, lunch, fika, and dinner, but also being there for me during the tough moments. Liv, thank you for the moments where we 'onze eieren kwijt konden'. Laurence, thank you for hosting me in your living room/co-working space. Daniël, thank you for the reinforcement learning talks and feedback. The 'SpaceGekkies', thank you for the fun game nights. To all my other friends, thank you for the laughs and companionship. Finally, Zhouping, thank you for your honest feedback on my work and my family for putting your confidence in me.

My time as a student has been a blast and has ended now. Thank you all for being there with me. Goodbye Delft University of Technology and Hello World.

Zhouxin Ge
Rotterdam, August, 2021

Contents

I

Scientific Article

End-to-End Hierarchical Reinforcement Learning for Online Adaptive Flight Control

Z.X. Ge *
Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands

Aircraft with disruptive designs have no high-fidelity and accurate flight models. At the same time, developing models for stochastic phenomena for traditional aircraft configurations are costly, and classical control methods cannot operate beyond the predefined operation points or adapt to unexpected changes to the aircraft. The Proximal Policy Option Critic (PPOC) is an end-to-end hierarchical reinforcement learning method that alleviates the need for a high-fidelity flight model and allows for adaptive flight control. This research contributes to the development and analysis of online adaptive flight control by comparing PPOC against a non-hierarchical method called Proximal Policy Optimization (PPO) and PPOC with a single Option (PPOC-1). The methods are tested on an extendable mass-spring-damper system and aircraft model. Subsequently, the agents are evaluated by their sample efficiency, reference tracking capability and adaptivity. The results show, unexpectedly, that PPO and PPOC-1 are more sample efficient than PPOC. Furthermore, both PPOC agents are able to successfully track the height profile, though the agents learn a policy that results in noisy actuator inputs. Finally, PPOC with multiple learned Options has the best adaptivity, as it is able to adapt to structural failure of the horizontal tailplane, sign change of pitch damping, and generalize to different aircraft.

Nomenclature

$s, s^R, a, \Delta a$	=	state, state reference, action and incremental action vector
ω, ω^k, Ω	=	single Option, single Option k and all Options
$\boldsymbol{\theta}, \boldsymbol{\theta_\pi}, \boldsymbol{\vartheta}, \boldsymbol{\vartheta_\beta}, \boldsymbol{\vartheta_\vartheta}$	=	parameter vector
$\eta, \gamma, \lambda, \epsilon$	=	learning rate, discount rate, variance parameter of GAE and clipping ratio
$J_x(\boldsymbol{\theta})$	=	objective function of x parameterized by parameter vector $\boldsymbol{\theta}$
V, V_Ω	=	state value function and state value function over Options
δ_t^V, r_{t+1}	=	temporal difference error using V and reward at the next time step
$\pi(\cdot \mid s)$	=	stochastic policy
$\mu(s, \boldsymbol{\theta}), \sigma(\boldsymbol{\theta})$	=	mean and standard deviation parameterized by parameter vector $\boldsymbol{\theta}$
ρ	=	importance sampling ratio
μ_π, μ_Ω	=	on-policy state distribution under π and Ω
$A_\pi, A_\omega, A_\Omega$	=	advantage function under policy π, intra-option policy ω and over Options Ω
Q_Ω, Q_U	=	option-value function and action-value in the context of (s, ω)
π_ω, β_ω	=	intra-option policy, and termination function of Option ω
$\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{R}$	=	state selector, state weight and action weight matrix
$x_i, \dot{x}_i, \dot{x}_i^R, F_i$	=	position, velocity, velocity reference and force of mass i
$q, q^R, \alpha, \theta, u, H, \delta_e$	=	pitch rate, pitch rate reference, angle of attack, pitch angle, airspeed, height and elevator deflection

I. Introduction

Recent developments of advanced reinforcement learning methods display promising characteristics that can enable model-independent, adaptive, and intelligent flight control for passenger carrying aircraft. Model-independent reinforcement learning can learn from scratch and thereby overcome the model information gap for novel aircraft designs. A type of passenger carrying aircraft that can benefit from this is aircraft with novel configurations. One category of

*Graduate Student, Faculty of Aerospace Engineering, Control and Simulation Division, Delft University of Technology

novel passenger carrying aircraft designs from the Urban Air Mobility (UAM) industry is the highly anticipated electric Vertical TakeOff and Landing (eVTOL) vehicles. For these designs, the UAM industry expects to lack detailed models and model information [1]. On the other hand, traditional aircraft configurations have a large backlog of research and detailed models. Still, creating an accurate model from scratch and developing models that incorporate stochastic phenomena (e.g., turbulence, birdstrike, and actuator failure) can be costly endeavors. In addition to model-independent flight control, reinforcement learning can enable adaptive flight control and replace classical control methods. Classical control methods limit an automatic pilot to be reliable and safe, only for predefined operating points. On the contrary, an adaptive flight control method allows for an automatic pilot to go beyond predefined operating points, to reduce the workload for a human pilot, and to improve flight safety during failure (e.g., actuator failure) [2].

A reinforcement learning method that allows for adaptive flight control is called Adaptive Critic Design (ACD) controllers. ACD's are learning algorithms tailored for optimal tracking of continuous-time control systems. These methods have successful implementations for helicopters [3] and fighter jets [4], but are limited to small state spaces and are dependent on representative simulation models. A family of adaptive methods that alleviate the need for an accurate dynamic model is called Incremental Approximate Dynamic Programming (IADP). An IADP method such as Incremental Dual Heuristic Programming (IDHP) has shown successful implementations for missiles [5] and jet aircraft [6] where the controllers did not require offline training. These IADP methods have proven to be effective in flight control for small continuous state and action space. Though these methods still have room for improvements and would profit from extending the controller design to a high-dimensional state space [6]. Also, the IADP methods would benefit from increased exploration capabilities to cope with unexpected changes to the environment and faster learning by requiring fewer samples (i.e., high sample efficiency) to learn a policy.

A field of research that can provide a better alternative and improvements for IDHP methods is the field of Deep Reinforcement Learning (DRL). DRL provides a foundation for research into high-performance methods that largely depend on Deep Neural Networks (DNN). A method of interest is the Proximal Policy Optimization (PPO) [7] which is a policy gradient reinforcement learning (PGRL) method that is widely used for its fast learning and exploration capabilities, which ultimately leads to finding good policies. Policy gradient methods are an intuitive approach for continuous control of systems and connect the IDHP and DRL methods. While using policy gradient methods for flight control, attention should be paid to a problem common to IADP methods. The problem being the 'curse of dimensionality', where the increase of dimensions will greatly reduce sample efficiency. While high sample efficiency is desirable for flight control as an aircraft will be able to adjust to new situations in a timely manner.

Hierarchical Reinforcement Learning (HRL) tackles the 'curse of dimensionality' by structuring and decomposing the state space of the agent's environment. Consequently, the agent is able to learn with fewer samples than an agent without Hierarchical Design [8]. Hierarchical Design can be expressed in a variety of ways by structuring and decomposing the underlying Markov Decision Process (MDP). The decomposition of the underlying dynamics of an RL problem is done by breaking it down into multiple 'activities' followed by a structure that links the multiple activities in a coherent Hierarchical Design. Three highly regarded Hierarchical Design methods are identified by [9] that all rely on the theory of Semi-MDP [10]. These methods are the Options [10], MAXQ [11], and Hierarchies of Abstract Machines (HAM) [12] framework. These HRL methods are frameworks that require the programmer to decompose the problem into 'activities'. This requires the programmer to have the domain knowledge to identify structures present in an environment. In adaptive flight control, this structure is not always known beforehand. In addition, it is preferred for the general applicability of the control design to have an HRL method that can learn to decompose the environment's hierarchical structure. A method that automatically decomposes the state space of the environment and does not need subgoals to be set by the programmer, and thus not requiring prior knowledge in the Hierarchical Design is the Proximal Policy Option-Critic (PPOC) [13]. PPOC combines the advantage of PPO and the benefits of HRL methods through the Option-Critic architecture [14] that provides the framework for end-to-end learned Options.

This paper contributes to the development and analysis of end-to-end learned Options for adaptive flight control by comparing the PPOC against PPO and PPOC with a single Option (PPOC-1). These two methods will be tested on an extendable mass-spring-damper system and jet aircraft models. The methods are evaluated on sample efficiency, reference tracking and adaptivity. In addition, it should be noted that analysis for online learning with a Hierarchical Design is not frequently encountered in literature. In this regard, this is one of the few works that contribute to the analysis for online adaptive flight control with a Hierarchical Design.

The paper is structured as follows. The theory behind PPOC is explained in section II, subsequently the agent design is presented in section III. The methodology is given in section IV. The results are found in section V, where the agent is evaluated for sample efficiency, reference tracking, offline training, offline adaptivity, online learning, online adaptivity, and transfer learning. The results of the adaptivity tests are discussed in section VI. The research is finalized with the

conclusions and recommendations in section VII.

II. Fundamentals

This section presents the theory required to understand and create the agent design in algorithm 1.

A. Reinforcement Learning

The Actor-Critic is a reinforcement learning framework that is generally used for continuous control problems. An Actor is responsible for selecting actions that maximize the expected total performance by receiving a feedback signal from the Critic. The Critic assesses if the action taken by the Actor has resulted in more or less value. The Actor-Critic is a policy gradient that updates its parameters using eq. (1). The parameters of the policy are updated using its current parameter estimate $\boldsymbol{\theta_t}$ and adding the gradient of the objective function's estimate multiplied by the learning rate η.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \nabla \hat{J}\left(\boldsymbol{\theta}_t\right) \tag{1}$$

The feedback signal from the Critic to the Actor comes in the form of a Temporal Difference (TD) error δ_t^V as described in eq. (2) which uses a parameterized value function $V\left(s_t, \boldsymbol{\vartheta_t}\right)$. The δ_t^V takes the difference between the current value and the future value using the reward of the next time step r_{t+1} and discounting future state value with γ.

$$\delta_t^V = r_{t+1} + \gamma V\left(s_{t+1}, \boldsymbol{\vartheta_t}\right) - V\left(s_t, \boldsymbol{\vartheta_t}\right) \tag{2}$$

The policy function is parameterized by a Gaussian distribution described by eq. (3) similar as [15] which makes it a stochastic policy. The advantage of this parameterization is that it allows the policy to explore in a natural fashion as opposed to deterministic policies that require exploration noise or signals to find good policies. In addition, the parameterization creates a continuous action space.

$$\pi(a \mid s, \boldsymbol{\theta}) \doteq \frac{1}{\sigma(\boldsymbol{\theta})\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s, \boldsymbol{\theta}))^2}{2\sigma(\boldsymbol{\theta})^2}\right) \tag{3}$$

The Actor-Critic Design presented here is a one-step TD method that forms a baseline for continuous and model-independent control. The drawback of one-step TD methods is the bias that is introduced through bootstrapping its value estimate.

B. Generalized Advantage Estimation

Instead of updating the value estimates by using bootstrapping by means of the TD error, an alternative approach in updating the policy gradient can reduce bias. The Generalized Advantage Estimation (GAE) [16] in eq. (4) allows for lower bias than one-step TD methods and reduce variance in policy gradient updates by using sampled trajectories. The low bias introduced through high γ allows the policy gradient to converge to better local optima than for higher bias value estimates. The reduction in variance through high λ allows for learning with fewer samples. The GAE estimation allows the state value-based δ_t^V defined in eq. (2) to estimate the advantage function. This is permitted, only if the estimation of the value function is determined by discounting with γ and based on a_t resulting from $\pi(a_t \mid s_t)$, then the δ_t^V is seen as an estimate of the advantage function of the action a_t [16]. Note that the TD error is used for estimating the advantage function and not for the value function.

$$A^{GAE(\gamma,\lambda)}(\delta_t^V) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \tag{4}$$

GAE estimates the advantage function that is used for the alternative expression of the policy gradient as defined in eq. (5). The intuition behind role of the adavantage function in eq. (5) is as follows. A positive advantage points the policy gradient in the direction of reward maximization, thereby increasing the probability of better-than-average actions. On a final note, as long as the timesteps required to predict the future value is lower than the duration of an episode, then the δ_t^V in GAE updates the policy function in an on-policy fashion. As a consequence, the agent should be able to learn in an online setting.

$$\nabla J(\boldsymbol{\theta}) = \sum_s \mu_\pi(s) \sum_a \nabla \ln \pi(a \mid s, \boldsymbol{\theta}) A_\pi(s, a), \tag{5}$$

C. Proximal Policy Optimization

The Proximal Policy Optimization (PPO) [7] alternates between sampling a trajectory and optimizing a 'surrogate' objective function as seen in eq. (6). The sampled trajectories will undergo an optimization process in order to have the best gradient updates. After a fixed amount of optimization iterations K using a minibatch M, the policy gets updated, and a new trajectory from the environment is taken for its next update. Note that $\hat{J}^{PPO}(\boldsymbol{\theta})$ in eq. (6) is not equal to $\nabla J(\boldsymbol{\theta})$. The objective function of PPO still needs to be differentiated.

$$\hat{J}^{PPO}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\min \left(\rho_t(\boldsymbol{\theta}) \hat{A}_t^{GAE}, \text{clip} \left(\rho_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon \right) \hat{A}_t^{GAE} \right) \right] \tag{6}$$

PPO's optimization process emulates the process of Trust Region Optimization as implemented in TRPO [17] by using a first-order iterative optimization method. As a consequence, PPO reduces the sample complexity while retaining the sample efficiency and guarantee of monotonic improvement seen in TRPO. The emulation is achieved by setting out a proximal region around a policy probability ratio $\rho_t(\boldsymbol{\theta}) = \frac{\pi(a_t|s_t,\boldsymbol{\theta})}{\pi(a_t|s_t,\boldsymbol{\theta}_{\text{old}})}$, where the samples gathered from an old policy is used to update to a new policy within the proximal region. Note that the use of $\rho(\boldsymbol{\theta})$ is the same as in the concept of importance sampling. The ratio $\rho(\boldsymbol{\theta})$ can become the lower bound of the objective function when the old policy has a higher advantage than the current policy.

The proximal region set by the clipping ratio ϵ ensures the stability of policy updates by not allowing for too large updates to be applied to the new policy. This is done by clipping $\rho_t(\theta)$ to the proximal region (1 - ϵ, 1+ ϵ). The intuition behind the clipping in eq. (6) is as follows. In case that the advantage is large and positive, then the upper region assures that the policy update is not too greedy for actions contributing to reward maximization. If the advantage is large and negative, then the lower region assures that the updates for actions that do not contribute to reward maximization are still having a reasonable probability of occurring.

D. Option-Critic Architecture

The Option-Critic architecture [14] automatically decomposes its problem domain into a set of temporally extended actions, which are called Options [10]. A way of viewing these Options is to see them as a set of sub-policies, with each having its own variable time scale in which the Option operates depending on its initiation state and termination condition. For the Options in [10], the initiation state and termination condition needs to be set by the programmer. In the case of the Option-Critic, the initiation set is omitted, and the termination condition is learned. The aforementioned sub-policies are also known as intra-option policies π_ω and the variable time scales are now set by their respective termination function β_ω. The goal of each Option is to maximize the expected return in the *current* task. The Options are selected and structured by the main policy called the policy over options π_Ω. The Option-Critic architecture learns these Options in an end-to-end fashion by applying the policy gradient theorem to the intra-option policies seen in eq. (7), termination functions seen in eq. (8). The policy learned with the intra-option policy gradient can be any reinforcement learning method as long as it meets the conditions of the intra-option policy theorem [14].

$$\nabla \hat{J}_{\pi_\omega}(\theta_t) = \sum_{s,\omega} \mu_\Omega \left(s, \omega \mid s_0, \omega_0\right) \sum_a \frac{\partial \pi_\omega(a|s,\boldsymbol{\theta})}{\partial \theta} Q_U(s,\omega,a) = \sum_{s,\omega} \mu_\Omega \left(s, \omega | s_0, \omega_0\right) \sum_a \nabla \ln \pi_\omega(a \mid s, \boldsymbol{\theta}) A_\omega^{GAE}(s,a) \tag{7}$$

$$\nabla \hat{J}_\beta(\vartheta_t) = -\sum_{s',\omega} \mu_\Omega \left(s', \omega | s_1, \omega_0\right) \frac{\partial \beta_\omega \left(s', \boldsymbol{\vartheta}\right)}{\partial \vartheta} A_\Omega^\beta \left(s', \omega\right) \tag{8}$$

The name Option-Critic stems from its similarities with the Actor-Critic Design, where the intra-option policies with their termination functions act as Actors that get selected by a higher level Actor called the policy over options. The intra-option policy, termination function, and policy over options get feedback from their respective Critics that estimate the action-value function in the context of a state-option pair Q_U and the advantage function of the policy over options A_Ω. The latter formulation of the intra-option policy in eq. (7) is taken from [13]. The alternative expression of the policy gradient performance function in eq. (5) allows us to rewrite the intra-option policy gradient in a form that the GAE function can be used. In addition, the advantage function in the termination gradient function follows directly from its derivation.

III. Agent design

The agent design used is elaborated where the specific settings for the agent are found in section IV.

A. Reward function

A quadratic cost-to-go function is used for continuous optimal tracking control. The implementation is similar to the one found in [6] and adapted for extra control on the magnitude of actions with the R matrix, as can be seen in eq. (9). The P and Q matrices allow for multiple states to be tracked and thus enable tracking for continuous MIMO systems.

$$r_{t+1} = r\left(s_t^R, s_{t+1}, \Delta a_t\right) = -\left[\boldsymbol{P}s_{t+1} - s_t^R\right]^T \boldsymbol{Q}\left[\boldsymbol{P}s_{t+1} - s_t^R\right] - \Delta a_t^T \boldsymbol{R}\Delta a_t \tag{9}$$

Where the state selector matrix P defines which states are selected for tracking, s_t being the state vector at time t, s_t^R being the reference signals, Q being the weighting matrix where the magnitude of each entry determines the priority of tracking with respect to each entry, R is the weighting matrix that sets the penalty for the magnitude of the action, Δa_t being action increment taken at time t.

B. Learning framework

The learning frameworks used for this research are the Proximal Policy Option Critic (PPOC)[13] and the PPO implementation of [18]. The PPOC method extends PPO with the Option-Critic in section II.D. As a result, both frameworks are on-policy methods, meaning that it updates its policy during an episode and learns without requiring episodes. In this paper, only the PPOC method is presented as the intra-option policy gradient function is the same as the policy gradient function of PPO. In fig. 1, the Option-Critic agent implemented with PPO as the intra-option policy is shown, where the differences between PPO and PPOC are indicated by the dashed red line. The algorithmic description of fig. 1 is given in algorithm 1 and follows directly from the formulas presented here and in section II. As seen in algorithm 1, PPOC does not use the temporal difference error to update the value estimate. As a consequence, this avoids introducing bias into the value estimate as is elaborated in section II.B. Instead, the value function uses a Mean Squared Error (MSE) function to update its estimation.

In fig. 2, the Actor Multi-Layer Perceptron (MLP) for PPOC is provided. There the intra-option policy is shown. The second head attached to this MLP is the policy over options but is not displayed for clarity. The termination function and the value function are combined into one MLP similar to the Actor MLP. In both MLP designs, the weights are shared, which increases sample efficiency. According to [19], this should be implemented in a more precise manner, though the termination function and the intra-option policy are independently estimated, meaning the functions have their own MLP and do not share weights which is a requirement for an Option-Critic.

The intra-option policies use the differential entropy of the Gaussian distribution H_{π_ω}, so the entropy bonus is only dependent on the σ and on the entropy hyperparameter. The entropy hyperparameter is a weight factor and determines the exploratory behavior of the policy. The same setup is used for the policy over options but instead uses the Shannon entropy H_{π_Ω}. The Shannon entropy defines the entropy for discrete random variables that corresponds to the mapping of states to options performed by the policy over options. All in all, the complete gradient of the objective function for the PPOC agent then becomes the following expressing that is given in eq. (10) and taken from section II, where S is the entropy coefficient in table 1, J_{π_Ω} is objective of the softmax policy, and J_{V_Ω} is the MSE.

$$\nabla J(\boldsymbol{\theta}_\omega, \boldsymbol{\vartheta}_\beta, \boldsymbol{\theta}_{\boldsymbol{\Omega}}, \boldsymbol{\vartheta}_\vartheta) = \nabla J_{\pi_\omega}^{PPO} + SH_{\pi_\omega} + \nabla J_\beta + \nabla J_{\pi_\Omega} + SH_{\pi_\Omega} + \nabla J_{V_\Omega} \tag{10}$$

C. Hyperparameters

In the preliminary analysis, a hyperparameter optimization tool called Optuna [20] was applied to the hyperparameters in table 1. The results indicate that hyperparameter optimization improves the sample efficiency of both agents but makes them less comparable. In order to stay within the scope of this research, the optimization and tuning of hyperparameters are left for detailed controller design. In table 1, the value for each hyperparameter is given. The same learning rate η is used for all required learning rates in algorithm 1.

D. Training strategy

The agent is training in an offline setting, where every episode terminates after 30 seconds of simulation time. The rewards are non-sparse. This allows the agent to collect a reward per time step. A sampling frequency of 100 Hz is employed as it is equal to the one used for the jet aircraft used for flight tests at the Delft University of Technology [21]. The training signal for the mass-spring-damper system is a sine signal with an amplitude of 5 m/s and a frequency of 0.2 Hz. The same signal is used for the aircraft system where the amplitude is equal to 5 $^\circ/s$.

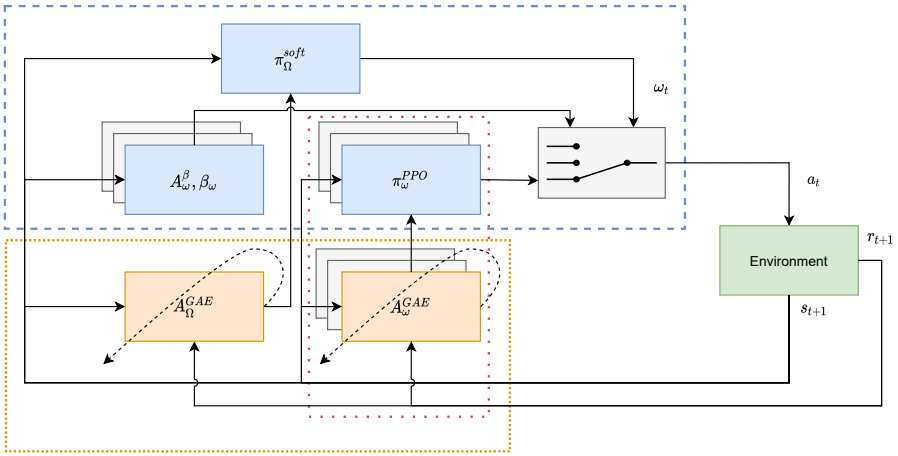

Fig. 1 Diagram of PPOC with three Options. The Actor components are in blue. The Critic components are in orange. A non-hierarchical Actor-Critic agent would only have the Actor and Critic component encircled with a red dashed line. The concept of Option selection is depicted by a switch with three contacts, where each contact resembles an Option. Only one Option is selected at a time and a new Option is selected according to π_Ω when the current Option terminates. Inside the GAE estimation, the δ_t is computed and passed to the GAE estimation and thus indirectly to the Actor.

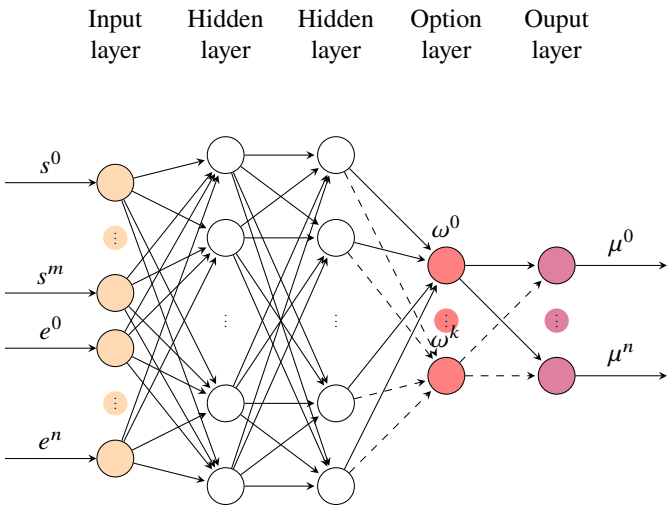

Fig. 2 Multi-Layer Perceptron topology of the Actor. The tracking error e_0 results from $Ps - s^R$. There are two hidden layers that each have 64 neurons and a hyperbolic tangent as an activation function. In this case, the intra-option policy ω_0 is activated, indicated with solid lines. The intra-option policy is equal to PPO. The policy over options is equal to a softmax, which selects which intra-option policy is activated. Each intra-option policy is estimating a mean. The standard deviation is state-independent. The MLP topology of the Critic has the same structure but outputs a state value and termination function with a sigmoid function over the output layer.

Algorithm 1 PPOC: Proximal Policy optimization with Option-Critic

Require:
agent hyperparameters $k, T, \epsilon, K, M, \eta_{\theta_\pi}, \eta_{\theta_\Omega}, \eta_{\vartheta_\beta}, \eta_{\vartheta_\vartheta}, S, \gamma, \lambda$
differentiable stochastic intra-option policy parameterization $\pi_\omega(a|s, \boldsymbol{\theta_\pi})$
differentiable termination function parameterization $\beta_\omega(s, \boldsymbol{\vartheta_\beta})$
differentiable stochastic policy over options parameterization $\pi_\Omega(\omega|s, \boldsymbol{\theta_\Omega})$
differentiable option-value function parameterization $V_\Omega(s, \omega, \boldsymbol{\vartheta_\vartheta})$

$s \leftarrow s_0$
$\omega \leftarrow \pi_\Omega(\omega \mid s, \boldsymbol{\vartheta_\vartheta})$ with softmax policy
while *episode* $\neq$ *terminal* **do**
  for iteration = $1, 2, \ldots$ **do**
    procedure RUN POLICY π_ω^{old} IN ENVIRONMENT FOR T TIME STEPS
      $a_t \leftarrow \pi_\omega(a_t \mid s_t, \boldsymbol{\theta_\pi})$
      take action a_t while in s_t and observe s_{t+1}, r_{t+1}
      if β_ω terminates in s_{t+1} **then**
        choose new ω_{t+1} according to softmax $\pi_\Omega(\omega_{t+1} \mid s_{t+1}, \boldsymbol{\vartheta_\vartheta})$
      end if
    end procedure
    procedure OPTIONS EVALUATION: COMPUTE ADVANTAGE ESTIMATES FOR T TIME STEPS
      $\delta_t \leftarrow r_{t+1} + \gamma V_\Omega(s_{t+1}, \omega_t, \boldsymbol{\vartheta_\vartheta}) - V_\Omega(s_t, \omega_t, \boldsymbol{\vartheta_\vartheta})$
      $\hat{A}_\Omega^{GAE}(s_t, \omega_t) \leftarrow \delta_t + \gamma\lambda\hat{A}_\Omega^{GAE}(s_t, \omega_t)$
      $\hat{A}_\Omega^{\beta}(s_t, \omega_t) \leftarrow \sum_{\omega=0}^{k} \pi_\Omega(\omega_t \mid s_t, \boldsymbol{\vartheta_\vartheta}) V_\Omega(s_t, \omega_t, \boldsymbol{\vartheta_\vartheta}) - V_\omega(s_t, \boldsymbol{\vartheta_\vartheta})$
    end procedure
    procedure OPTIONS IMPROVEMENT
      for $\omega = \omega^0, \omega^1, \ldots, \omega^k$ **do**
        $\theta_\pi^{old} \leftarrow \theta_\pi$
        for K optimizer epochs with minibatches M **do**
          $\theta_\pi \leftarrow \theta_\pi + \eta_{\theta_\pi} \frac{\partial \hat{J}^{\mathrm{PPO}}(\theta_\pi)}{\partial \theta_\pi}$
          $\vartheta_\beta \leftarrow \vartheta_\beta - \eta_{\vartheta_\beta} \frac{\partial \beta_\omega(s_{t+1}, \boldsymbol{\vartheta_\beta})}{\partial \vartheta_\beta} \hat{A}_\Omega^{\beta}(s_t, \omega_t)$
          $\theta_\Omega \leftarrow \theta_\Omega + \eta_{\theta_\Omega} \frac{\partial \log \pi_\Omega(\omega_t|s_t, \boldsymbol{\vartheta_\vartheta})}{\partial \theta_\Omega} \hat{A}_\Omega^{GAE}(s_t, \omega_t)$
          $\vartheta_\vartheta \leftarrow \vartheta_\vartheta - \eta_{\vartheta_\vartheta} \frac{\partial (G - V_\Omega(s_t, \omega_t, \boldsymbol{\vartheta_\vartheta}))^2}{\partial \vartheta_\vartheta}$
        end for
      end for
    end procedure
  end for
end while

Table 1 The hyperparameter values are taken from [18]

Hyperparameter symbol	Name	Value
k	Number of Options	variable per experiment
T	Actor batch	256
ϵ	Clipping ratio	0.2
K	Optimization epochs	10
M	Optimization batchsize	64
η	Learning rate	0.005
S	Entropy coefficient	0.01
γ	Discount factor	0.99
λ	Variance parameter of GAE	0.95

IV. Methodology

The PPOC and PPO method are tested on two types of systems: a mass-spring-damper (MSD) system and aircraft. In testing, a distinction is made between an offline and online setting, where offline means that the agents make use of episodes during training. Online means that the agents are learning without requiring episodes. First, in section IV.A, three settings of the PPOC method are tested on an extendable MSD system in an offline setting. The goal of this setup is to analyze the offline adaptivity and sample efficiency of the PPOC agent compared to PPO. Secondly, in section IV.B, the proposed method is tested on two aircraft systems. The goal of the second setup is to have a proof-of-concept for adaptive flight control with PPOC and display the benefits of using learned Options in an online setting.

A. Mass-spring-damper system

Different environments are set up using an extendable mass-spring-damper-N system (MSD-N) to find differences between these agents. The letter N is replaced by an integer defining the number of serially coupled MSD systems, fig. 3 provides an example of an MSD-3 system. The need for testing on multiple environments stems from the variation in algorithm performance across environments. In general, the best-performing algorithm across multiple environments is not always clear. A way to quantify overall algorithm behavior and to have good interpretability is to have an environment that can be increased in complexity. For this reason, an extendable MSD system is used. The MSD simulation model can be found [22].

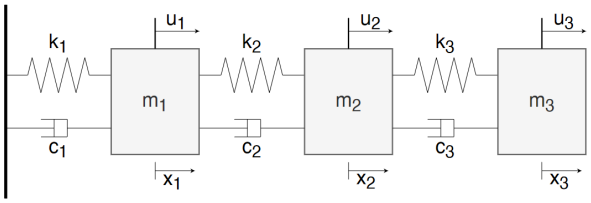

Fig. 3 MSD-3, a mass-spring-damper system with three masses. The image is from [23].

Only two experiments are shown in this paper. These experiments emulate longitudinal control where experiment 1 (EXP1) has a small state space and experiment 2 (EXP2) has a large state space. EXP1 is a single-mass-spring-damper system with the following state, reference, and action vector eq. (11) with the following P, Q, and R matrices in eq. (12). EXP2 is a three-mass-spring-damper system with the following state, reference, and action vector eq. (13) with the following P, Q, and R matrices in eq. (14). For the other experiments, the reader is referred to the technical report [22]. In the technical report, the EXP2 is renamed to EXP3, and experiments regarding offline adaptivity and sample efficiency that emulate lateral control can be found.

$$s = \begin{bmatrix} x_1 & \dot{x}_1 \end{bmatrix}^T \quad s = \begin{bmatrix} \dot{x}_1^R \end{bmatrix}^T \quad a = \begin{bmatrix} F_1 \end{bmatrix}^T \tag{11}$$

$$\boldsymbol{P} = \begin{bmatrix} 0 & 1 \end{bmatrix} \quad \boldsymbol{Q} = \begin{bmatrix} 1 \end{bmatrix} \quad \boldsymbol{R} = \begin{bmatrix} 0 \end{bmatrix} \tag{12}$$

$$s = \begin{bmatrix} x_1 & x_2 & x_3 & \dot{x}_1 & \dot{x}_2 & \dot{x}_3 \end{bmatrix}^T \quad s^R = \begin{bmatrix} \dot{x}_1^R \end{bmatrix}^T \quad a = \begin{bmatrix} F_1 \end{bmatrix}^T \tag{13}$$

$$\boldsymbol{P} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \boldsymbol{Q} = \begin{bmatrix} 1 \end{bmatrix} \quad \boldsymbol{R} = \begin{bmatrix} 0 \end{bmatrix} \tag{14}$$

For both experiments, the first mass m_1 needs to follow a sine signal. Adaptivity is tested by changing the internal dynamics of the mass-spring-damper system. The internal dynamics are changed by switching the sign of the damping constant. For EXP1, the damping constant is changed from $c_1 = 3$ to $c_1' = -1$. For EXP2, it changes from $c_1 = 3$ to $c_1'' = -1.5$, as the MSD-3 remains stable for $c_1' = -1$. The sign change makes the system unstable and provides insight into how well the agents can adapt to abrupt changes to dynamics. Adaptivity is measured by the recovery success rate. The recovery success rate is defined by the number of recoveries after changing the internal dynamics at half the training time. The recovery success rate is measured at the end of the training, whereas the nominal success rate is measured before the change. An agent has recovered when the reward is within the range of -50 and 0. Sample efficiency is determined by observing the initial slope of the training curve.

B. Aircraft system

The aircraft system is a simulation model using general linearized equations of motion describing aircraft motion. The equations of motion are taken from a technical report published by the Faculty of Aerospace Engineering of the Delft University of Technology [24]. The stability and control derivatives are taken from the same technical report for two different aircraft, a jet aircraft during a cruise and a different jet aircraft during an approach. The use of linearized models provides reliable results for small deviations from the linearization point and using a high sampling frequency. The simulation is run with a sampling frequency of 100 Hz, see section III.D.

A more elaborate control design is required for flight control than for the MSD system. From initial flight control experiments, the output resulting from the agent had high frequency and high gain control. These characteristics are both not desirable for flight control as they will decrease the durability of the actuator and introduce unwanted vibrations. A working strategy from [25] that allows for more control on the agent's output is to treat the agent's outcome $\boldsymbol{a}^{\text{agent}}$ as an angular velocity and by limiting the velocity with an upper $\Delta\boldsymbol{a}_{max}$and lower bound $\Delta\boldsymbol{a}_{min}$ of $1\text{deg}/s$ and $-1\text{deg}/s$ in eq. (15). Then by integrating using eq. (16) the angular velocity, the actuator deflection gets smoothed out over time, thus reducing the high frequency. In addition, to discourage the agent from yielding high gain control, a penalty is given for large action increments.

$$\Delta\boldsymbol{a} = \Delta\boldsymbol{a}_{\text{min}} + \left(\boldsymbol{a}^{\text{agent}} + 1\right) \frac{\Delta\boldsymbol{a}_{\text{max}} - \Delta\boldsymbol{a}_{\text{min}}}{2} \tag{15}$$

$$\boldsymbol{a}_t = \boldsymbol{a}_{t-1} + \Delta\boldsymbol{a}_t \tag{16}$$

As a result, the following state, reference, and action vector in eq. (17) are used with the P, Q, and R matrices in eq. (18) for the flight control experiments. The flight control design employs two PID controllers that ultimately provide a q^R to the PPOC agent, as seen in fig. 4. In addition, the aircraft's thrust is controlled through an airspeed controller embedded in the aircraft.

$$s = \begin{bmatrix} \alpha & \theta & q & \delta_e \end{bmatrix}^T \quad s^R = \begin{bmatrix} q^R \end{bmatrix}^T \quad a = \begin{bmatrix} \Delta\delta_e \end{bmatrix}^T \tag{17}$$

$$\boldsymbol{P} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \quad \boldsymbol{Q} = \begin{bmatrix} 2 \end{bmatrix} \quad \boldsymbol{R} = \begin{bmatrix} 1 \end{bmatrix} \tag{18}$$

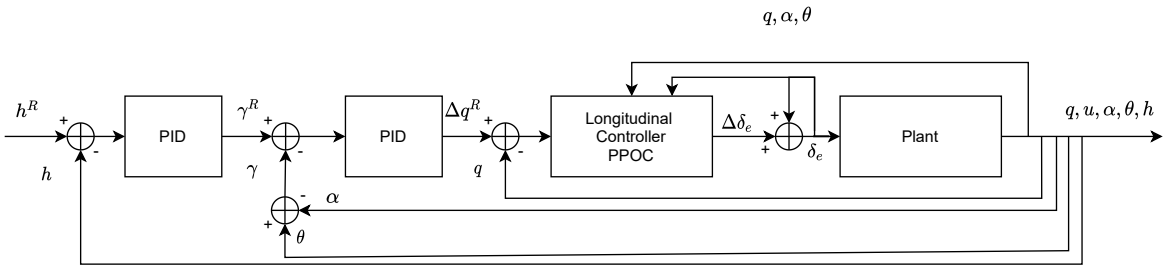

Fig. 4 Control Diagram of PPOC for longitudinal flight control.

Similar to the MSD system, the agent for the aircraft system is trained with a sine signal. The sine signal is the same. Although, the adaptivity is tested differently. First, the agent's online performance is tested by tracking a height profile, where the initial height is taken at $2000m$, but has no physical relevance as the aircraft are linearized for their respective linearization conditions. The maneuvers and duration for the height profile are provided in table 2. Secondly, online adaptivity is tested through three different tests while following a height profile. The first test is a partial loss of horizontal tailplane at $t = 50s$, more in section V.B.4. The second test is a sign switch of Cm_q at $t = 50s$ and is equivalent to the c-test used for the MSD system, see section V.B.5 for the results. The last test transfers the agents to a Boeing 747-100 during an approach at $t = 0s$, more in section V.B.6.

The PPOC agent's input-output mapping is assessed after offline training in section V.B.1 and after online learning in section V.B.3. The assessment will use a linear signal described by eq. (19). The linear signal is passed through the agent acting as pitch rate or pitch rate reference error. The remaining states are set to zero during the testing. The

Table 2 Overview of the duration of each segment for the height profile.

segment	level	climb	level	descend	level
time $[s]$	5	150	30	45	40

input-output mapping is obtained by fixing the agent's weights after learning. The output of the agent is evaluated for five seconds, where each time step is equal to $0.01s$.

$$f(t) = \frac{2}{180}\pi t - \frac{5}{180}\pi \tag{19}$$

V. Results

The results are built up in two parts. First, in section V.A, the proposed method's adaptivity is tested in an offline setting. Secondly, in section V.B, the proposed is trained offline on an aircraft system, and then the method's adaptivity is tested in an online setting. For the definition of the online and offline setting see the introduction of section IV. Finally, the results from this section will be discussed on a higher level in section VI.

A. Mass-spring-damper system

The results concerning offline adaptivity is presented in section V.A.1 and offline sample efficiency in section V.A.2. All agents are trained for 2400 seconds while using the same offline training setup in section III.D. The training time is the result of a trial and error process to get good policies.

1. Evaluation of offline adaptivity

A table of the two experiments in the offline training setting is given in table 3, where a total of 20 runs were executed for each agent. The nominal and recovery phase are within one run. The results in the table should be read as follows for PPO and EXP1. A total of 18 runs are completed, and 12 of those 18 runs satisfy the definition of a nominal run given in section IV.A. For these experiments, the PPOC agent shows that it is better at recovering from abrupt changes to its dynamics than PPO, as the PPOC agent has higher success rates during the recovery phase. The recovery success rates are especially high for EXP2 for PPOC-4, whereas PPO is not able to recover within the desired reward threshold. The results suggest that PPOC is more adaptive due to its Options. An intuition that arises from PPOC's behavior is that the learned Options stabilize the response to changes and learning of new behavior through switching between Options, more about this in section V.B. On a final note, the PPO method also has a lower nominal success rate, as the agent keeps exploring in a quite aggressive manner. For EXP1, it even has two failed runs due to this exploration behavior.

Table 3 Overview of the success rates for offline adaptivity. The experiments are performed for PPO and PPOC-k, where k defines the amount of Options used for the experiment.

EXP	PPO		PPOC-2		PPOC-4		PPOC-8	
	nominal	recovery	nominal	recovery	nominal	recovery	nominal	recovery
1	12/18	00/18	19/20	13/20	20/20	20/20	19/20	17/20
2	11/20	00/20	20/20	04/20	20/20	19/20	17/20	18/20

2. Offline sample efficiency and adaptivity

The PPO agent is the most sample efficient method during the nominal training phase, whereas the PPOC agent is the most sample efficient method during the recovery phase. This observation is supported by looking at the initial slopes for the first and second half of the learning curve in section V.A.1. To have a better understanding of the sample efficiency, we zoom in on the results from section V.A.1. The learning curves for EXP1 and EXP2 for the best PPOC agent and PPO are given in fig. 5. Here the PPO agent shows high sample efficiency for both experiments, whereas the

PPOC agent is lower. On the contrary, the PPOC agent shows higher sample efficiency for both experiments during the recovery phase, whereas the PPO agent is not recovering for any of the two experiments. These results show that the PPOC agent is more adaptive to changes to dynamics and recover from that with a higher sample efficiency than for a framework that is not using learned Options.

In continuation of zooming in on the results, in fig. 6 the learning curves for all agents for only EXP2 are shown. Here it is seen that learning Options is beneficial for offline training, though learning too many options impacts the sample efficiency during the nominal phase. In addition, a tradeoff must be made between higher nominal sample efficiency or a higher sample efficiency during recovery of unexpected change to internal dynamics.

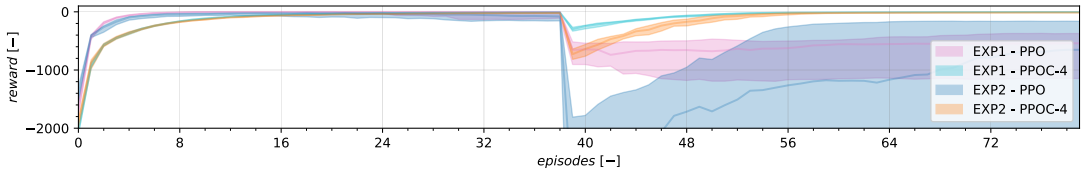

Fig. 5 Offline training curves of PPOC-4 and PPO for EXP1 and EXP2. Each shaded area indicates the interquartile range over 20 runs. The line in the shaded area is the mean line over 20 runs.

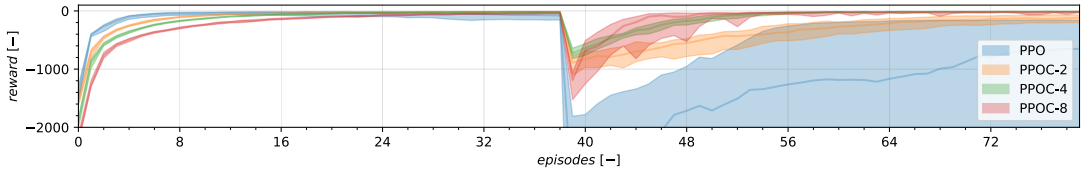

Fig. 6 Offline training curve of all agents for EXP2. Each shaded area indicates the interquartile range over 20 runs. The line in the shaded area is the mean line over 20 runs.

B. Aircraft system

In this section, the PPOC agent with multiple Options (PPOC-2) is compared against PPOC with a single Option (PPOC-1) to study the effects and benefits of using multiple Options more closely. Both agents are trained for 3500 seconds while using the same offline training setup in section III.D. The training time is the result of a trial and error process to get good policies. After training, the agents are put into an online learning setting where the agents need to track a height profile in a variety of conditions as explained in section IV.B.

1. Evaluation of offline training

The training curves in fig. 7 show that PPOC-1 is consistent for the first eight episodes, and then in a few runs, PPOC-1 starts exploring. PPOC-2 has a noticeable variation amongst the 20 runs for the first 30 episodes. This indicates that having multiple Options in this system will result in a higher variance due to multiple intra-option policies that are not prolonged long enough, as PPOC-1 is – in essence – a single intra-option policy that will never be terminated. Eventually, both agents converge to a local optimum, as seen after the 30 episodes mark. In addition, the median line of both agents is comparable, though the PPOC-2 agent is slightly lower. This is consistent with the offline sample efficiency for the nominal phase in section V.A.

The figures in the left column of fig. 8 show that the agent is capable of following a sine signal without requiring any excitation signal. The interrelations between the states are consistent with longitudinal aircraft motion and the states are small enough. Thus reliable inferences can be made about the performance of PPOC for flight control. In the right column, the state-option response of the policy over options π_Ω is shown. The figures show multiple circles overlapping each other. This observation is consistent with the cyclic nature of the sine signal. The colors in the overlapping circles indicate what intra-option policy π_ω are selected by π_Ω for the states. When matching the state-option-action figures to the time traces then this will give insight into how π_Ω behaves for all states over time. In addition, it provides insight into

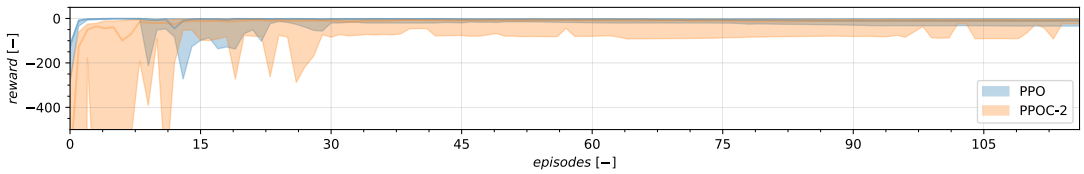

Fig. 7 The learning curves of PPOC-1 and PPOC-2. Each shaded area indicates the interquartile range over 20 runs. The line in the shaded area is the mean line over 20 runs. A longer simulation time will not give better performance as both agents have reached a local optimum.

how the continuous actions are composed out of the two Options. So the observations of the α-δ_e graph tell us what Options are selected in the q-δ_e figure. The α-δ_e graph indicates that the π_Ω quite consistently selects either of two Options for actions that are increasing and decreasing. In the case of the α-δ_e graph, π_Ω selects with a higher probability ω^1 for increasing α and selects ω^0 for decreasing α. An emphasis on probability should be placed as the π_Ω still selects ω^0 for increasing α. This means that the actions are selected from two different but similar probability distributions that, on average, create a sine signal and a thick band around the output signal, as seen in the time trace of δ_e.

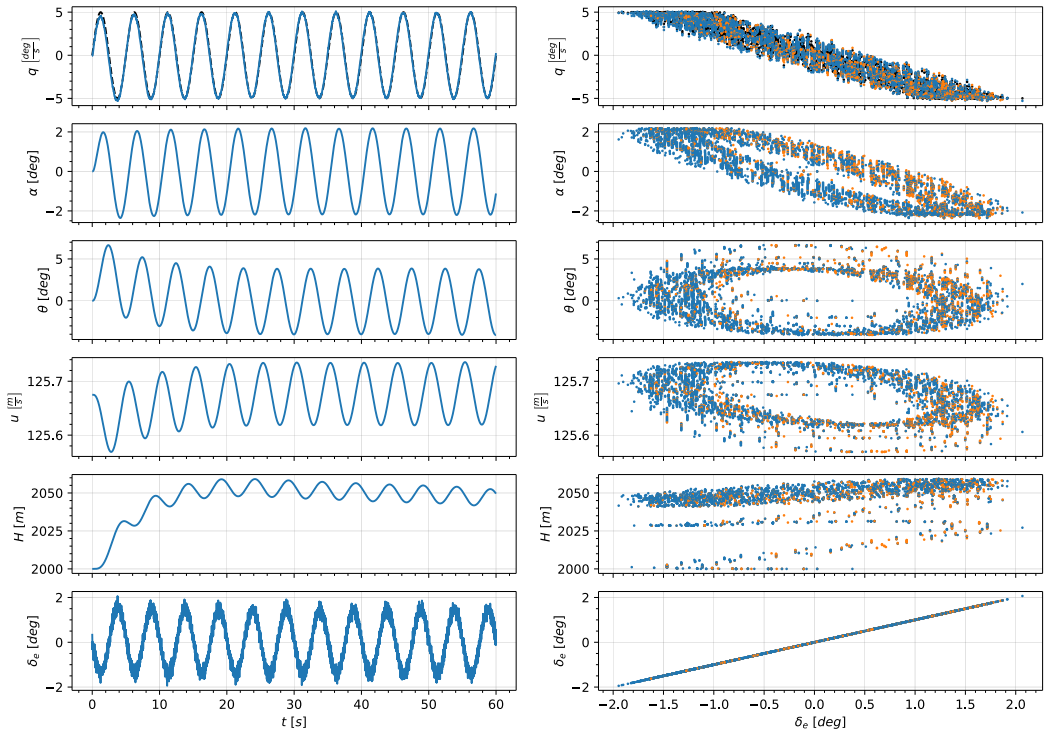

Fig. 8 In the left column, the time traces during offline training evaluation are seen. The right column displays the states to δ_e figures resulting from the same samples used for the left column. The selected Options are in color. The Options ω^0 and ω^1 are indicated in blue and orange, respectively. The black dashed line is the pitch reference tracking signal. The agent's performance is evaluated for 60 seconds, displaying that it generalizes and sustains the aircraft's dynamics for longer durations.

To study the agent's response more closely, we zoom in on the agent's response to a single state. For this a different approach is applied that results in fig. 9, where the approach is explained in section IV.B. The slope of agent's response for q matches the slope of the ellipse seen in fig. 8, where the agent increases δ_e for decreasing q. This corresponds to the expected aircraft motion as a negative elevator deflection will increase the pitch rate. Looking at the agent's

response for q^e then for a positive error, the aircraft should pitch down using a positive elevator deflection. Vice versa for a negative error. This behavior is seen in fig. 9. The slopes of the δ_e-q graphs in fig. 9 have the opposite sign, this indicates that the agent has learned the right relation between the q and q^e which is $q^e = q - q^R$. All in all, this means that the agent has had sufficient training and learned a policy that should be able to track a reference signal, more about this in the next section.

In fig. 9 the δ_e-q graph shows a slight offset from the zero error line for ω^0 and ω^1. This observation corresponds to the initial tracking error in fig. 8, which is not equal to zero and results in the agent to immediately compensate for this. This means that the intra-option policies have learned a bias towards positive q, where it needs a higher q to obtain an increase in δ_e. So the agent is less sensitive for q around zero when compared to q_e. Another observation, the Ω for q compensates for the offset created by the intra-option policies by alternating between these policies. In addition, symmetry in the option selection is observed in the right column. This indicates that Options are chosen for specific absolute values of q^e and correspond to the observations made for the α-δ_e figure in fig. 8.

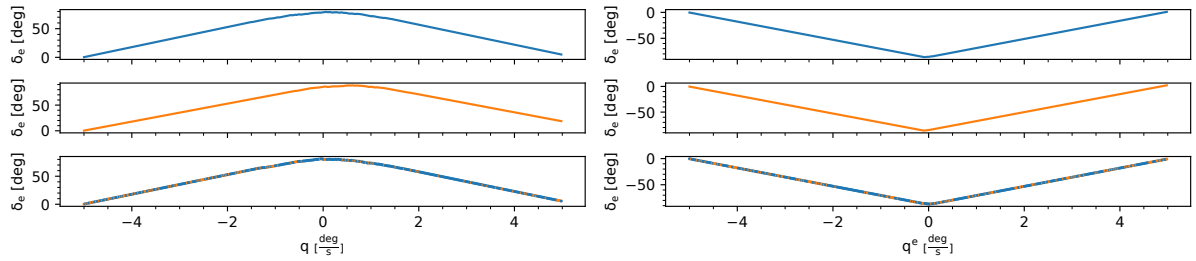

Fig. 9 Input and output mapping of PPOC-2 after offline training. The first, second, and third rows are the i/o mapping of ω^0, ω^1, and Ω, respectively. The input-output mapping shows successive additions of incremental actions, so the mapping does not represent the agent's instantaneous response to a specific value of a state in a single time step. The mappings relate increasing or decreasing q and q^e to increasing or decreasing δ_e.

2. Evaluation of online learning

The PPOC agent is able to track a reference signal not seen during offline training as seen in fig. 10. The PPOC agent follows a height reference tracking signal using the approach in section IV.B. Though the offline learned behavior is still visible,as the agent's actions start small and continue in a sinusoidal manner. The agent's ability to generalize for a different reference signal stems from its ability to re-adjusts its parameters in a stable manner as is seen in fig. 11. There it is seen that relatively large weight adjustments are made at $t = 0s$ and $t = 150s$, as the agent needs to adjust itself to the change in the reference signal. The stable updates are attributed to the policy iterations that take place every $2.56s$ seen at $t = 0s$. So the weights are adjusted after every 256 time steps. This number corresponds to the Actor batch hyperparameter, and this is how PPO updates its weights. So PPOC is an online adaptive and thus is able to generalize for an unseen reference signal.

The input-output mapping of the agent after online learning in fig. 12 is changed when compared to mapping after offline training. The agent has shifted its response to q to the left, which is a consequence of a prolonged climb. The shift to the left for q causes the agent to initiate a negative δ_e for lower q, causing the aircraft to pitch up earlier. Thus the agent has learned to pitch up when the aircraft's pitch rate is a small negative value. In addition, the agent has learned to sustain, on average, a zero pitch rate during the climb. Finally, looking at the $\delta_e - q^e$ graph, it can be seen that the agent developed a preference for using ω^1 when the error is negative. This indicates that the ω^1 policy contains a mapping that is better at handling negative errors.

In addition, the π_Ω starts to choose Option ω^0 more consistently. The H-δ_e figure in fig. 10 shows that Option ω^0 is chosen more frequent during level flight. When matching the position in the time traces for level flight and correlating this to the u-δ_e and θ-δ_e figures. Then it confirms that Option ω^0 is the preferred learned policy for level flight. This observation is confirmed by counting the selected options per segment of the flight profile. In table 4 it is seen that ω^0 is three to ten times more preferred than ω^1. A learned PPOC agent can have a learned policy over options that are using multiple Options or a single Option. In this case, the PPOC agent has learned a policy over Options that uses multiple options but prefers ω^0 over ω^1 as is seen in table 4. In the remainder of this paper, we will see the benefits of using multiple learned Options over a single Option agent.

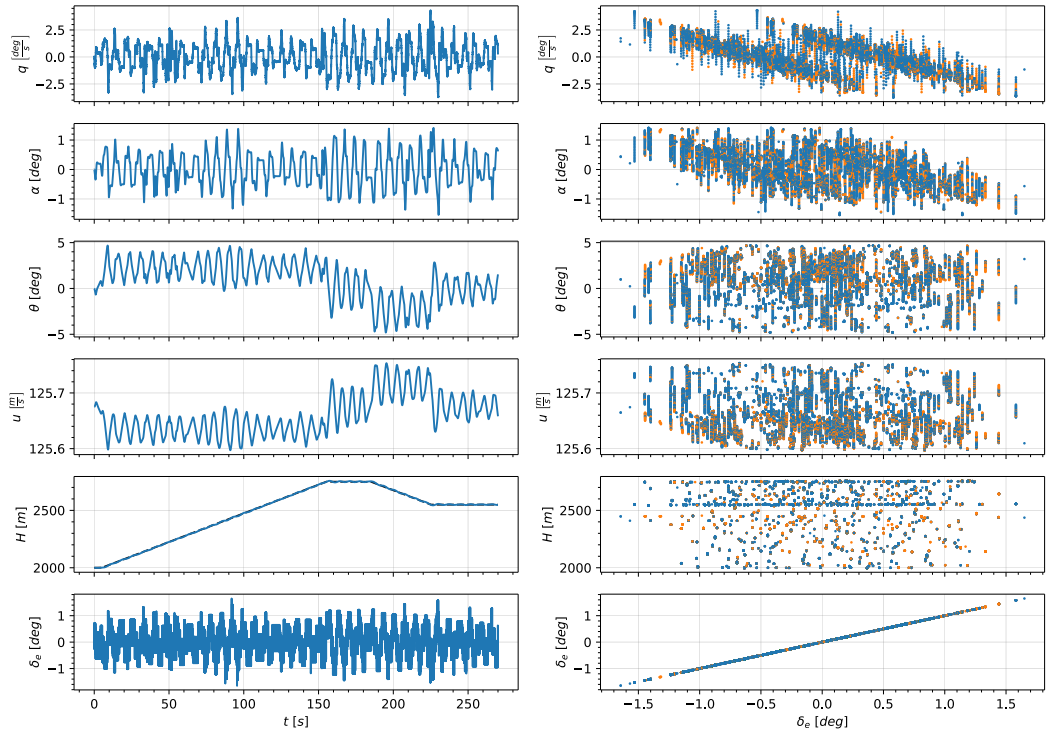

Fig. 10 **The left column shows the time traces during online learning. The right column displays the states to δ_e figures. The right column displays the states to δ_e figures resulting from the same samples used for the left column. The selected Options are in color. The Options ω^0 and ω^1 are indicated in blue and orange, respectively. The black dashed line is the height tracking reference signal.**

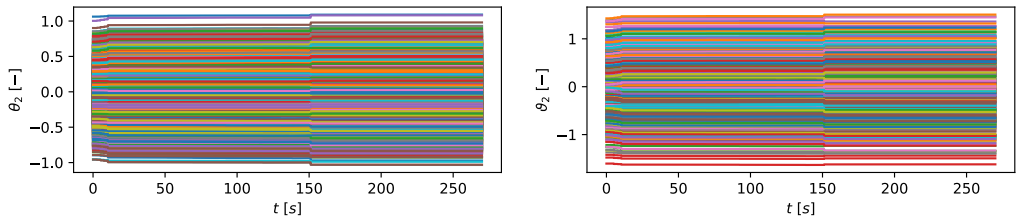

Fig. 11 **The time traces of the weights are shown for the last layer of the Actor (on the left) and the Critic network (on the right).**

Table 4 **An overview of Option selection per segment during online learning.**

selection	level	climb	level	descend	level
ω^0	419	7683	2127	3441	3587
ω^1	81	6317	873	1059	413
total	500	15000	3000	4500	4000

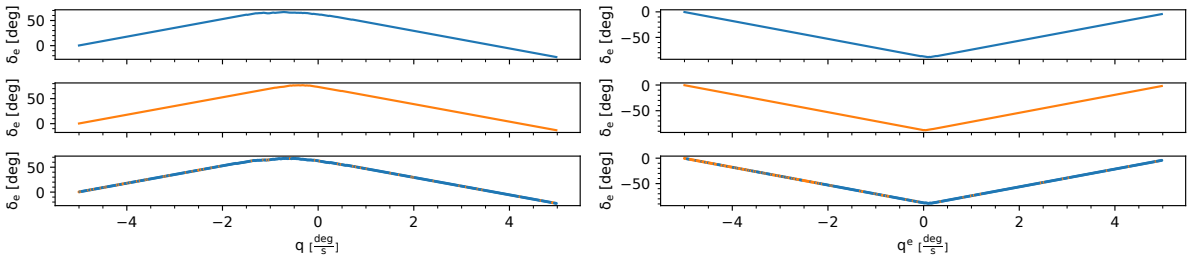

Fig. 12 Input and output mapping of PPOC-2 after online learning. The first, second, and third rows are the i/o mapping of ω^0, ω^1, and Ω, respectively. The input-output mapping shows successive additions of incremental actions, so the mapping does not represent the agent's instantaneous response to a specific value of a state in a single time step. The mappings relate increasing or decreasing q and q^e to increasing or decreasing δ_e.

3. Evaluation of PPOC-2 and PPOC-1 online learning

The PPOC-2 agent is more consistent than its single Option counterpart, as it has a higher success rate and the standard deviation remains constant, as is seen in fig. 13. Though, PPOC-1 has a much lower standard deviation. The actions produced by the PPOC-1 agent grow over time and explains the low success rate, as it becomes unstable over time. Thus using an agent using more than one Option allows for reliable reference tracking. This is also seen for the offline adaptivity results in section V.A.1.

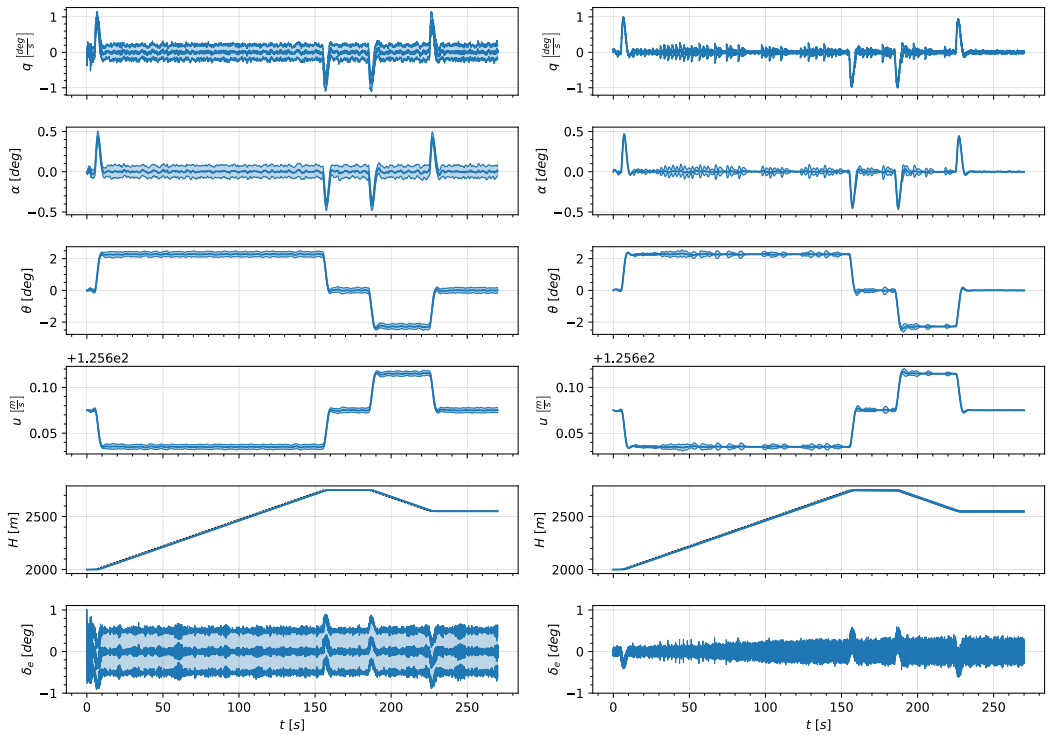

Fig. 13 Nominal flight operation. Left column PPOC-2 with success rate 99/100. Right column PPOC-1 with success rate 60/100. The bold line in the center is the average taken over 100 runs. The shaded area is the mean plus or minus a single standard deviation taken over 100 runs.

4. Evaluation of PPOC-2 and PPOC-1 during a partial loss of horizontal tailplane

A bird strike can cause considerable damage to the aircraft's structure. In case a bird hits the horizontal tailplane, it can reduce control effectiveness. A structural failure to the aircraft's horizontal tailplane is simulated by reducing the pitch damping and elevator control effectiveness by 70%. The derivatives related to q and δ_e are reduced by 70%.

The PPOC-2 agent shows that it is better at handling a bird strike on its horizontal tail than using a single Option, as seen in fig. 14. Another observation is that the standard deviation for both agents decreases after the structural failure at $t = 50s$. Though at $t = 155s$, agents both require a larger elevator deflection than for the nominal case in section V.B.3 due to the decrease in control effectiveness. For the PPOC-1 agent, this leads to unstable control behavior and leads to a low success rate. Thus using more than one Option allows for high resilience to unexpected change in pitch damping and control effectiveness. As a result, this leads to sustained and stable flight control during a partial loss of horizontal tailplane.

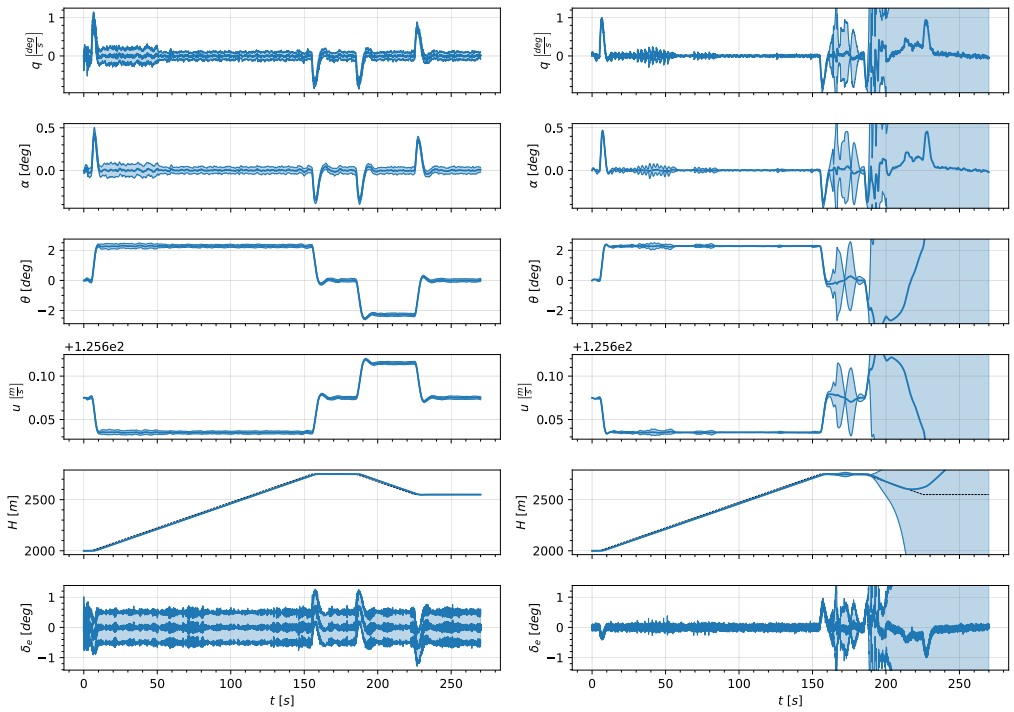

Fig. 14 Partial loss of horizontal tailplane at $t = 50s$. Left column PPOC-2 with success rate 99/100. Right column PPOC-1 with success rate 68/100. The bold line in the center is the average taken over 100 runs. The shaded area is the mean plus or minus a single standard deviation taken over 100 runs.

5. Evaluation of PPOC-2 and PPOC-1 with sign switch of C_{m_q}

The aircraft response to an abrupt change to internal dynamics of the aircraft is studied by changing the sign of the pitch damping C_{m_q} at $t = 50s$, where the pitch motion becomes negatively damped, resulting in increasing oscillatory flight. The results in fig. 15 show that both agents are tolerant to a negative pitch damping. The negative pitch damping did not result in a failure of the aircraft as the success rates and time traces are similar to the nominal case. Though the time traces for both agents are noisier, that is a consequence of the negative pitch damping that forces the agent to counteract the undamped pitch motion. Accordingly, the δ_e signal is less pronounced at each transition of the segment when compared to fig. 13. So no additional benefit of using multiple Options is seen here.

6. Evaluation of PPOC-2 and PPOC-1 during a transfer on a different aircraft

A flight controller that generalizes to different flight conditions or different aircraft allows for fast flight control design. A general flight controller can act as a baseline design. In the context of reinforcement learning flight controllers,

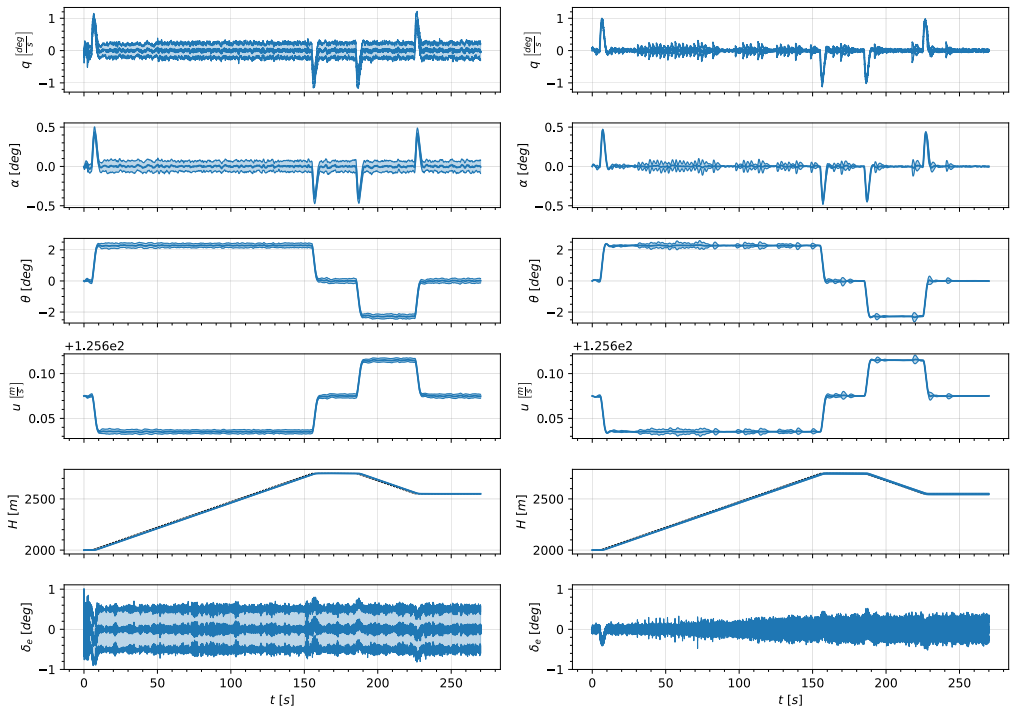

Fig. 15 Sign switch. Left column PPOC with success rate 98/100. Right column PPOC-1 with success rate 62/100. The bold line in the center is the average taken over 100 runs. The shaded area is the mean plus or minus a single standard deviation taken over 100 runs.

this can greatly reduce training time and expand to more complex cases. To this extend, the agents are transferred to an aircraft that is very different in its dimensions and class. The learned agents are transferred from a small jet aircraft to a large jet aircraft in an approach configuration instead of a cruise. The main differences in the aircraft's dynamics are in the center of gravity, pitch damping, and elevator control dynamics. This requires the agents to adapt their previously learned policies to output larger elevator deflections.

The PPOC-2 agent successfully transfers its learning to a different aircraft as seen in fig. 16, where the agent uses its gained experience in a new context. This can be explained by extrapolating the findings in section V.B.3, where the agent learns to control the large jet aircraft by switching between the learned Options that stabilize the online learning. On the contrary, the PPOC-1 agent is not able to transfer this experience. Around $t = 50s$, the agent loses control for all runs as it cannot switch between policies. Although PPOC-2 is successful in transfer learning, it shows reduced tracking performance, as it lags behind the height tracking reference. All in all, the results in fig. 16 show adaptive control and promise that it allows for easy transfer to non-linear dynamics.

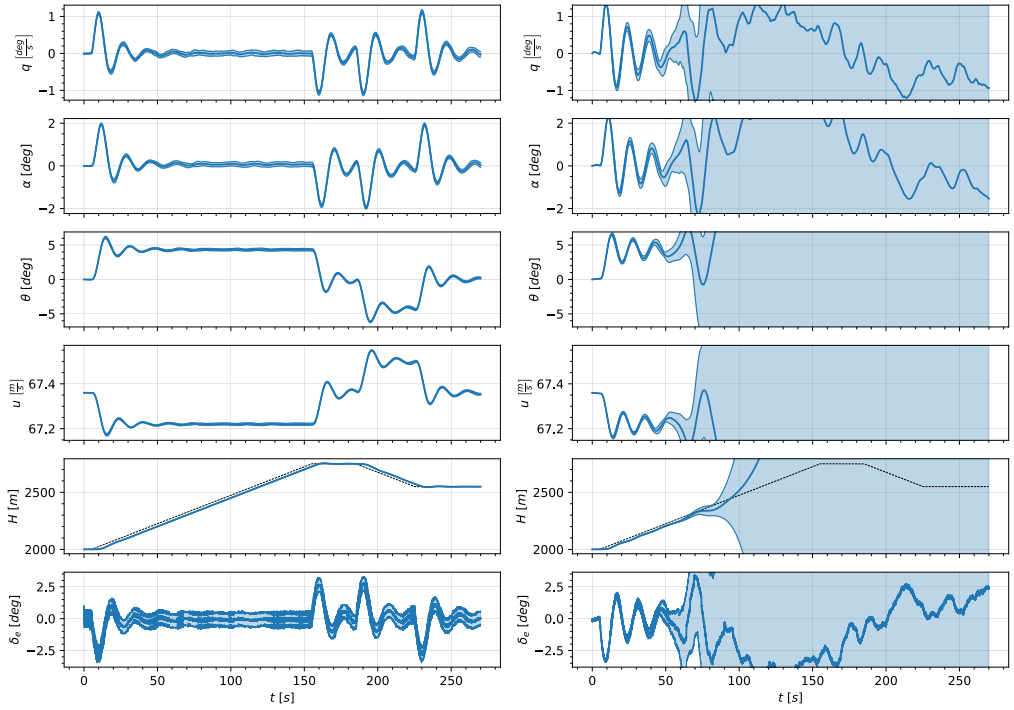

Fig. 16 Transfer learning of an linearized aircraft model to linearized model of Boeing 747-100 during approach. Left column PPOC with success rate 100/100. Right column PPOC-1 with success rate 57/100. The bold line in the center is the average taken over 100 runs. The shaded area is the mean plus or minus a single standard deviation taken over 100 runs.

VI. Discussions

Surprisingly the anticipated sample efficiency gain through hierarchical reinforcement learning is not realized for the offline training for both the MSD system and aircraft model. PPOC using a single Option and PPO – a single policy method – are the most sample efficient, and the sample efficiency of learned Options decreases with increasing Options. This is explained by the fact that each additional learned Option requires more parameters to be learned. An additional explanation might be in the fundamental difference in the classical application of Options, where hand-crafted intra-option policies have prior knowledge embedded in them. Embedding prior knowledge, in general, allows reinforcement learning methods to be more sample efficient. Summarizing the results with regards to adaptivity, PPOC with multiple learned Options give higher success rates than PPO for offline adaptivity. In addition, PPOC with multiple learned Options leads to higher success rates than PPOC-1 during tracking of a height profile while having a

structural failure of the horizontal tailplane, sign change of pitch damping, and generalizes to a different aircraft. PPOC is able to extend its learning to a different aircraft without requiring extra offline training. This shows that having more learned Options result in the ability to generalize over different aircraft. In addition, it allows for a stabilizing effect during online learning and adaptivity. Thus multiple learned Options allow for more complex control behavior and be adaptive in a reliable manner.

In regards to reference tracking, the PPOC agent is able to follow the reference signals for both MSD and aircraft systems in the offline and online setting. Though PPOC is not displaying the desired actuation behavior even after applying the strategy for integrating the actions, see section IV.B. Two sources that contribute to the noisy output of the PPOC agent are identified. The first source is the interaction between the termination function and policy over options causes high frequency switching between Options. As a consequence, PPOC learns a high-frequency control policy. The second source is the underlying stochastic policy method. This is seen for the PPOC-1 agent, where the high-frequency output is the result of the Gaussian distribution used for sampling the actions.

The observed high-frequency output can be reduced with the following three approaches. The first is a simple solution where the high-frequency output can be reduced by passing it through a low pass filter, though this will introduce lag and less control by the agent. Another more fundamental approach is to have more control over the stochasticity of the stochastic policies by making the entropy variable with a temperature variable. The last proposed approach is in the direction of having more control over the switching induced by the termination function. For the latter, there is an existing method that adds a deliberation cost [26] to the termination gradient. The deliberation cost lets the agent switch less frequently by incurring a cost for switching.

VII. Conclusions and recommendations

PPOC is capable of learning how to control an aircraft's inner loop dynamics, though it requires offline training before it can be applied in an online setting. Still, the method does not require any model information and solely uses samples resulting from interactions with the model. Multiple on-policy learned Options can enable a deep reinforcement learning method to have height reference tracking and online adaptivity in the cases of structural failure of the horizontal tailplane and sign change of pitch damping. In addition, PPOC is able to generalize to a different aircraft by means of transfer learning. Although, the method does not provide sample efficiency benefits over its non-hierarchical counterpart. Unfortunately, PPOC is not suitable for flight control due to its high-frequency control input to the actuators as it leads to reduced durability of the actuators and unwanted vibrations. More research into controlling stochasticity of the stochastic policies and learning of the termination function can lead to more control over the agent's output. Consequently, it should then render the PPOC method suitable for adaptive flight control of novel and traditional aircraft configurations, where ultimately further research is required that includes the effects of actuator dynamics, non-linear aircraft dynamics, and flight tests.

References

[1] Silva, C., Johnson, W., Antcliff, K. R., and Patterson, M. D., "VTOL urban air mobility concept vehicles for technology development," *2018 Aviation Technology, Integration, and Operations Conference*, 2018, pp. 1–16. https://doi.org/10.2514/6.2018-3847.

[2] Syed, S., Khan, Z. H., Salman, M., Ali, U., and Aziz, A., "Adaptive flight control of an aircraft with actuator faults," *2014 International Conference on Robotics and Emerging Allied Technologies in Engineering, iCREATE 2014 - Proceedings*, 2014, pp. 249–254. https://doi.org/10.1109/iCREATE.2014.6828374.

[3] Enns, R., and Si, J., "Helicopter trimming and tracking control using direct neural dynamic programming," *IEEE Transactions on Neural Networks*, Vol. 14, No. 4, 2003, pp. 929–939. https://doi.org/10.1109/TNN.2003.813839.

[4] Van Kampen, E., Chu, Q. P., and Mulder, J. A., "Continuous adaptive critic flight control aided with approximated plant dynamics," *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2006*, Vol. 5, No. August, 2006, pp. 2989–3016. https://doi.org/10.2514/6.2006-6429.

[5] Zhou, Y., Van Kampen, E. J., and Chu, Q. P., "Incremental approximate dynamic programming for nonlinear adaptive tracking control with partial observability," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 12, 2018, pp. 2554–2567. https://doi.org/10.2514/1.G003472.

[6] Heyer, S., Kroezen, D., and van Kampen, E., "Online adaptive incremental reinforcement learning flight control for a cs-25 class aircraft," *AIAA Scitech 2020 Forum*, Vol. AIAA 2020-1844, American Institute of Aeronautics and Astronautics,

Reston, Virginia, 2020. https://doi.org/10.2514/6.2020-1844, URL http://resolver.tudelft.nl/uuid:38547b1d-0535-4b30-a348-67ac40c7ddcchttps://arc.aiaa.org/doi/10.2514/6.2020-1844.

[7] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv: 1707.06347*, 2017, pp. 1–12. URL http://arxiv.org/abs/1707.06347.

[8] Nachum, O., Lee, H., Gu, S., and Levine, S., "Data-efficient hierarchical reinforcement learning," *Advances in Neural Information Processing Systems*, Vol. 2018-Decem, No. Nips, 2018, pp. 3303–3313.

[9] Barto, A. G., and Mahadevan, S., "Recent Advances in Hierarchical Reinforcement Learning," *Discrete Event Dynamic Systems*, Vol. 13, No. 1–2, 2003, p. 41–77. https://doi.org/10.1023/A:1022140919877, URL https://doi.org/10.1023/A:1022140919877.

[10] Sutton, R. S., Precup, D., and Singh, S., "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *ARTIFICIAL INTELLIGENCE*, Vol. 112, No. 1-2, 1999, pp. 181–211. https://doi.org/10.1016/S0004-3702(99)00052-1.

[11] Dietterich, T. G., "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition," *Journal of Artificial Intelligence Research*, 2000. https://doi.org/10.1613/jair.639.

[12] Parr, R., and Russell, S., "Reinforcement learning with hierarchies of machines," *Advances in Neural Information Processing Systems*, 1998, pp. 1043–1049.

[13] Klissarov, M., Bacon, P.-L., Harb, J., and Precup, D., "Learnings Options End-to-End for Continuous Action Tasks," *arXiv preprint arXiv: 1712.00004*, 2017. URL http://arxiv.org/abs/1712.00004.

[14] Bacon, P.-L., Harb, J., and Precup, D., "The option-critic architecture," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31, 2017.

[15] Degris, T., Pilarski, P. M., and Sutton, R. S., "Model-Free reinforcement learning with continuous action in practice," *Proceedings of the American Control Conference*, 2012, pp. 2177–2182. https://doi.org/10.1109/acc.2012.6315022.

[16] Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P., "High-dimensional continuous control using generalized advantage estimation," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016, pp. 1–14.

[17] Schulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P., "Trust region policy optimization," *32nd International Conference on Machine Learning, ICML 2015*, Vol. 3, 2015, pp. 1889–1897.

[18] Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y., "Stable Baselines," https://github.com/hill-a/stable-baselines, 2018.

[19] Riemer, M., Cases, I., Rosenbaum, C., Liu, M., and Tesauro, G., "On the Role of Weight Sharing During Deep Option Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, No. 04, 2020, pp. 5519–5526. https://doi.org/10.1609/aaai.v34i04.6003, URL https://ojs.aaai.org/index.php/AAAI/article/view/6003.

[20] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M., "Optuna: A Next-Generation Hyperparameter Optimization Framework," *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Association for Computing Machinery, New York, NY, USA, 2019, p. 2623–2631. https://doi.org/10.1145/3292500.3330701, URL https://doi.org/10.1145/3292500.3330701.

[21] Van den Hoek, M., de Visser, C., and Pool, D., "Identification of a Cessna Citation II model based on flight test data," *Advances in Aerospace Guidance, Navigation and Control*, Springer, 2018, pp. 259–277.

[22] Ge, Z. X., "End-to-End Hierarchical Reinforcement for Adaptive Flight Control," Tech. rep., Delft University of Technology, 2021.

[23] Buysscher, D., "Safe Curriculum Learning For Linear Systems With Unknown Dynamics In Primary Flight Control," Tech. rep., Delft University of Technology, 2021.

[24] Mulder, J. A., Van Staveren, W. H. J. J., Van Der Vaart, J. C., De Weerdt, E., De Visser, C. C., In 't Veld, A. C., and Mooij, E., "Lecture Notes AE3202 Flight Dynamics," Tech. rep., Delft University of Technology, 2013.

[25] Dally, K., "Deep Reinforcement Flight Control Learning for," Tech. rep., Delft University of Technology, 2021.

[26] Harb, J., Bacon, P. L., Klissarov, M., and Precup, D., "When waiting is not an option: Learning options with a deliberation cost," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018, pp. 3165–3172.

II

Preliminary Report (Previously graded under AE4020)

1

Introduction

Breakthroughs in the development of autonomous systems caused a resurgence in interest in black box optimization methods such as reinforcement learning. One of the recent historical breakthroughs is the AlphaGo Zero algorithm [57] that learns from scratch how to play the game of Go and then defeat the world champion of Go. Interestingly, the key technology – reinforcement learning – enabling this breakthrough was already invented in the 1950s by Arthur Samuel [60], and in 1961 his algorithm was able to defeat a national checker champion player in the USA [38]. In that same year, the work of Donald Michie embodied in project MENACE, described a simple trial-and-error learning system using matchboxes to learn from scratch how to play tic-tac-toe.

So the development of advanced reinforcement learning methods has come a long way and the current methods display promising characteristics that can enable intelligent, autonomous, and safe flight control for passenger carrying aircraft. In addition, it might provide a solution for the model information gap for novel aircraft designs that are becoming highly feasible through the expected near-term availability of mature technology. One category of novel aircraft designs from the Urban Air Mobility (UAM) industry, is the highly anticipated electric Vertical TakeOff and Landing (eVTOL) vehicles. For these novel aircraft configurations, the UAM industry expects to lack detailed model information [53]. On the other hand, traditional aircraft configurations have a large backlog of research and detailed models. Still, creating an accurate model from scratch and developing models that incorporate stochastic phenomena (e.g., turbulence, birdstrike, and actuator failure) can be costly endeavors.

A traditional approach to handle unexpected events is to design an automatic pilot that comprises multiple PID controllers that are tuned for specific operating points [31]. If an unexpected event occurs that is not in the region of the predefined operating points, then the automatic pilot will hand over control to the human pilot who will manually control the aircraft. In the case of a failure that results in loss of control, a human pilot might not have obtained the required situational awareness – as a consequence of a high workload – to get the aircraft under control in a timely manner. Thus a need is identified for a more advanced automatic pilot that can go beyond the predefined operating points without requiring a detailed dynamic model. And at the same time, the advanced automatic pilot – just like the human pilot – must adapt to this unexpected situation in a timely manner.

Adaptive flight control allows for an automatic pilot to go beyond predefined operating points, to reduce the workload for a human pilot, and to improve flight safety during failure (e.g., actuator failure) [63]. A method that alleviates the need for an accurate dynamic model and allows for adaptive flight control is called Incremental Non-linear Dynamic Inversion (INDI). For INDI, the overall controller becomes significantly less dependent on accurate aircraft model properties, especially regarding the aerodynamic derivatives [52] [58]. For aerodynamic derivatives, the industry expects to have limited knowledge for UAM vehicles [34]. Another family of model-independent adaptive controllers is Adaptive Critic Design (ACD) controllers. ACD's are reinforcement learning algorithms tailored for optimal tracking of continuous-time control systems. These methods have successful implementations for helicopters [18] and fighter jet [65]. A family of methods that combine the advantages of both INDI and ACD is called Incremental Approximate Dynamic Programming (IADP). An IADP method such as Incremental Dual Heuristic Programming (IDHP) has shown successful implementations for missiles [80] and jet aircraft [25] where the controllers did not require offline training.

IADP methods have proven implementations in flight control for small continuous state and action space

but would profit from extending the controller design to a high-dimensional state space [25]. Also, the IADP methods would benefit from increased exploration capabilities to cope with unexpected changes to the environment and faster learning by requiring fewer samples (i.e., high sample efficiency) to learn a good policy. A field of interest that can improve IDHP methods is the field of Deep Reinforcement Learning (DRL). DRL provides a wide body of research into high-performance methods that largely depend on Deep Neural Networks (DNN), where AlphaGo Zero [57] is a product of this field. Researchers in DRL conduct extensive research in high-dimensional problems and exploration to improve adaptability in a class of reinforcement learning methods called policy gradient reinforcement learning (PGRL) methods. Where the Actor-Critic Design provides a bridge between IADP methods and DRL methods with Actor-Critic Design.

Actor-Critic Design, in general, provides a good baseline for continuous control, though when applied to adaptive flight control, special attention should be paid to sample efficiency. High sample efficiency allows a flight control method to adjust to new situations in a timely manner. A field that is known for its Hierarchical Design approach to tackling the challenge of obtaining high sample efficiency in high-dimensional control space is Hierarchical Reinforcement Learning (HRL). Where the control space – a state and/or action space – is decomposed into smaller solution spaces and linked to a hierarchical structure. As a consequence, a reinforcement learning algorithm with Hierarchical Design can result in faster learning for high-dimensional problem domains [40]. In the field of Hierarchical Reinforcement Learning, three highly regarded methods were identified [5] that were common in their reliance on the theory of Semi-MDP [59]. These methods are the Options [59], MAXQ [16], and Hierarchies of Abstract Machines (HAM) [43] framework. A fourth framework that was not in this review of Barto [5] – as it does not rely on the Semi-MDP but on the MDP as it is not a temporal abstraction but a state abstraction – is Feudal Reinforcement Learning [13]. These HRL methods are frameworks that require the programmer to decompose the problem into 'activities'. This requires the programmer to have the domain knowledge to identify structures present in an environment. In adaptive flight control, this structure is not always known. In addition, it is preferred for the general applicability of the control design to have an HRL method that can learn to decompose the environment's hierarchical structure.

In this research, the author combines the advantages of Actor-Critic Design with the advantages of Hierarchical Design. Thereby, the author investigates Hierarchical Policy Gradient Reinforcement Learning (HPGRL) methods that learn from scratch. With the proposed method, the author satisfies the following research objective:

Contribute to the development of a novel model-independent and adaptive controller for a continuous, high-dimensional, partially observable, and stochastic problem domain by investigating a hierarchical policy gradient reinforcement learning flight controller for a fixed-wing aircraft that enables sample efficient flight recovery from unexpected changes to aircraft dynamics.

The research proposal is given in chapter 2, where the research objective is supported by the main research question and sub-research questions. In chapter 3 requirements for flight control and fundamentals of PGRL techniques are given. The main findings of the thesis are written in article form in part I. A review of the state-of-the-art PGRL methods and HPGRL methods is provided in chapter 4. In chapter 5 a comparison is made between a PGRL and an HPGRL technique on a mass-spring-damper system, where special attention is given to adaptivity, sample efficiency, and reference tracking. Finally, this research concludes in chapter 6 and with a set of recommendations in chapter 7.

2

Research proposal

Following from the introduction, where the research gap is identified, the research proposal is presented here. First, the field of research is provided in section 2.1. Secondly, an overview of the research objective and questions is given in section 2.2.

2.1. Field of Research

The modern field of reinforcement learning dates back to the late 1980s by combining three separate research areas. A research area was studying the concept of learning through trial-and-error and originated from the psychology of animal learning. Another field was trying to solve various optimal control problems by using value functions and dynamic programming. Yet another field was looking into temporal-difference methods which interrelated these two independent fields [61].

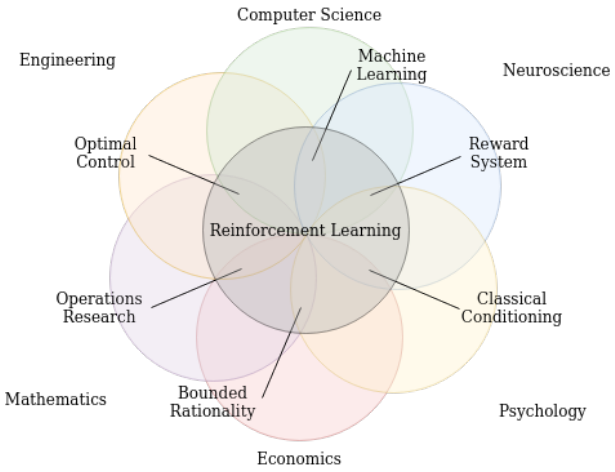

Figure 2.1: An overview of the broad research field of modern reinforcement learning with its intersections of other research areas. According to this diagram the research area Optimal Control is the intersection of Computer Science, Engineering, Mathematics and Reinforcement Learning. This image is adapted from David Silver [54].

Modern reinforcement learning has seen an expansion and wide adoption in different existing research areas, that is in neuroscience, psychology, economics, mathematics, engineering, and computer science [54]. The area of interest for this research is where mathematics, engineering, and computer science intersect, as shown in fig. 2.1. The areas intersect at optimal control and machine learning. For this research, the author will extract knowledge and techniques from a wide body of literature available in machine learning and apply this to the field of optimal tracking control for aircraft.

Within the field of reinforcement learning, a distinction is made between model-dependent and model-independent techniques which can be value-based, policy-based, or both as shown in fig. 2.2. The non-hierarchical policy gradient methods in this research will be mainly methods with an Actor-Critic Design, which are model-independent policy gradient agents that are suitable for continuous state-action space.

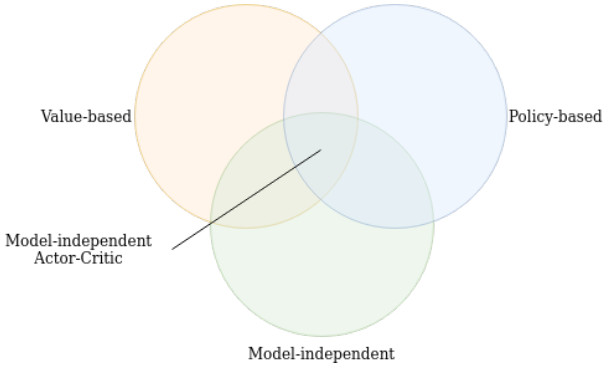

Figure 2.2: An overview of algorithms in modern reinforcement learning that are characterized by four main aspects: value-based, policy-based, model-independent and model-dependent. These aspects are not mutually exclusive. The image is inspired from [55].

2.2. Research objective and questions

The research objective is translated into four research questions. The research questions each have several supporting sub-questions. The four research questions and their sub-questions are formulated as such that they collectively answer the main research objective. The set of research questions are presented below.

Research objective

Contribute to the development of a novel model-independent and adaptive controller for a continuous, high-dimensional, partially observable, and stochastic problem domain by investigating a hierarchical policy gradient reinforcement learning flight controller for a fixed-wing aircraft that enables sample efficient flight recovery from unexpected changes to aircraft dynamics.

Research questions

1. What are the requirements for a fixed-wing aircraft flight controller?
   (a) What are the classical control loops?
   (b) What are the state and action spaces for a fixed-wing aircraft?
   (c) What are the characteristic dynamic motions of a fixed-wing aircraft?
   (d) How should a reinforcement learning problem for flight control be formulated?
2. What are the current state-of-the-art reinforcement learning methods that are suitable for flight control?
   (a) What is the most suitable state-of-the-art policy gradient reinforcement learning technique?
   (b) What is the most suitable state-of-the-art hierarchical policy gradient reinforcement learning technique?
3. What are the benefits of the proposed hierarchical policy gradient technique over the proposed non-hierarchical policy gradient technique on a mass-spring-damper system when exposed to unexpected changes?
   (a) What is the performance regarding adaptivity?
   (b) What is the performance regarding sample efficiency?
   (c) What is the performance regarding reference tracking?
4. How should the proposed hierarchical policy gradient method be implemented for flight control of a fixed-wing aircraft?
   (a) What is the performance regarding adaptivity, sample efficiency, and reference tracking when exposed to unexpected changes?
   (b) How can sample efficiency be improved?
   (c) How can reference tracking be improved?

3

Literature study part I: Fundamentals

Following from the research proposal in chapter 2, the theoretical foundation for this research is set in this chapter. Where the design practices of an automatic flight control system is elaborated and comes with requirements that is translated into the characteristics that a suitable reinforcement learning method should adhere to. Consecutively, the fundamentals of reinforcement learning and policy gradient methods will be given. Ultimately, this chapter aims to set the theoretical of this research by implicitly answering the first research question and its sub-research question as stated in section 2.2 and restated here:

1. **What are the requirements for a fixed-wing aircraft flight controller?**
   (a) *What are the classical control loops?*
   (b) *What are the state and action spaces for a fixed-wing aircraft?*
   (c) *What are the characteristic dynamic motions of Cessna Citation I?*
   (d) *How should a reinforcement learning problem for flight control be formulated?*

In answering this research question, an overview of the requirements for flight control resulting from a problem analysis of flight control for a CS-25 aircraft is provided in section 3.1. This is followed by the fundamentals of reinforcement learning methods in section 3.2. This chapter puts an emphasis on policy gradient reinforcement learning in section 3.3 and concludes in section 3.4.

3.1. Problem analysis of flight control for CS-25 aircraft

The requirements for flight control of a fixed-wing CS-25 aircraft are presented in this section with the aim to implicitly answer the first research question and its sub-research questions. After this section, a reinforcement learning controller can be referred to as an RL agent or agent. The aircraft dynamics described by the state-space systems will be called the plant or environment of the agent, and the control signal is equivalent to a action from the agent.

Classical control loops and timescales

In aircraft systems, a human pilot is an intelligent, adaptive, and versatile control system. A human pilot can assess written, visual and spoken information from displays, weather, terrain, air traffic, air traffic control, and crew. From these information sources and a specific task, a pilot plans a set of actions to be executed in a particular order as to go from point A to point B in a safe manner. A set of actions related to controlling an aircraft are manual control and automatic control. The pilot's actions related to automatic control are facilitated through a mode control panel, which allows a pilot to set reference parameters for the autopilot system.

In general, a human pilot can set reference parameters for heading angle, airspeed, pitch attitude, altitude, and vertical speed. These reference parameters are also linked to the flight management system. The flight management system is the center of the aircraft onboard avionics architecture and also interfaces with the autopilot. The set of reference parameters fixed by the human pilot are also communicated to the flight management system, which computes a trajectory based on the flight plan, the prevailing conditions, and

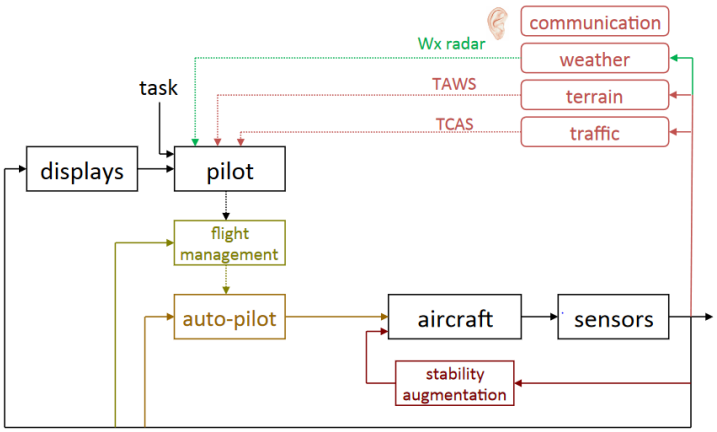

Figure 3.1: A high level overview of control loops in aircraft employed in commercial aviation and placement of the human-pilot, flight managements system, and autopilot. Image is adapted from [8].

the optimized operation of the aircraft. From these computations, the flight management system can also select a reference airspeed, altitude, and engine power settings during all phases of flight for the autopilot. So an autopilot has two sources for reference parameters, where the values set by the human pilot has the highest priority. A high level overview of the placement of the human pilot, flight management system, and autopilot is given in fig. 3.1.

The task of an autopilot is to optimally track a reference signal or hold a target parameter. In fig. 3.2 a block diagram of an autopilot for lateral control is given that can hold a heading angle. This autopilot design assures that an aircraft will maintain a reference heading angle using the aileron as actuator for turning and maintaining a zero side slip angle using the rudder. In this design two controllers are implemented and one damper to stabilize the aircraft's Dutch Roll. In this controller design the controller parameters e.g., PID values are set by a gain schedule for specific flight conditions and/or using interpolation techniques to encompasses more flight conditions. In case that an aircraft's response cannot be controlled by these techniques a loss of control can occur. During loss of control , in general, the autopilot will hand over control to the human pilot. In these cases the pilot can be in a high stress situation and this might influence the pilot's situational awareness and work load. For safety-critical situations it is desired to ease the work load of the pilot as to enable flight recovery.

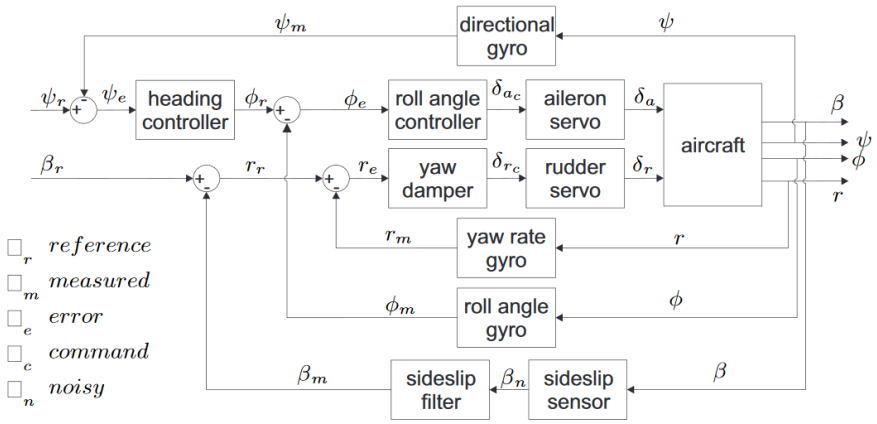

Figure 3.2: Example of two subsystems combined into one, but handled as two subsystems with their own single input and multiple outputs. This is a heading angle hold mode, this decoupled design can be used for a mass damper spring system with two masses of which the two masses need to follow a displacement reference each. Image is adapted from [66].

So in case of loss of control, there is a need for an adaptive autopilot. If adhering to the autopilot design given in fig. 3.2 the adaptive autopilot will replace the heading and roll angle controller. Preferably the adaptive autopilot can replace these two separate Single Input Single Output (SISO)controllers by one adaptive

controller as to reduce overhead in the controller architecture and enable more control for the adaptive controller. Another possibility among many, can be an adaptive Multi Input Multi Output (MIMO)controller that replaces the roll rate controller and yaw damper that resulting in an adaptive controller that has control over four degrees of freedom of the aircraft.

Another autopilot design concerning the longitudinal motion also using a cascaded control system structure is presented in fig. 3.3. It has a feedback loop for each cascade and is augmented with a two-time-scale motion in the closed loop system such that the controller dynamics is a singular perturbation with respect to the airplane dynamic. In fig. 3.2 stability of fast motion transients is maintained by selection of controller parameters for the inner loop. While the slow motion behavior of Euler angles have the desired transients performance for tracking tasks. In order to support this an overview of characteristic modes of a Cessna Ce500 'Citation' with their period and time to damp half the amplitude in table 3.1.

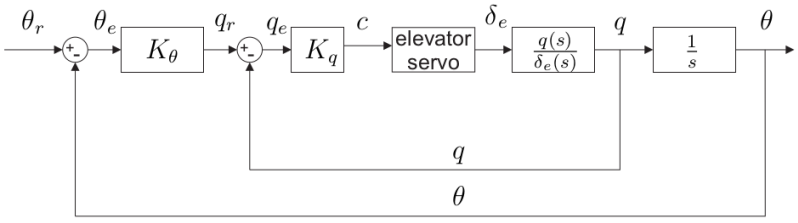

Figure 3.3: An example of a pitch attitude hold autopilot with a two-time scale motion in a closed loop system. Image is adapted from [66].

State and action space

A general state space representation of linearized aircraft dynamics was taken from [39]. The state space representation was derived from the following assumptions. The deviations from the initial flight condition are small enough to permit linearization of the nonlinear aircraft dynamics. A flight condition is characterized by airspeed and altitude. The aircraft is in steady, straight, symmetric flight. No aerodynamic coupling exists between the symmetric and the asymmetric degrees of freedom, as long as the deviations and disturbances remain small.

From these assumptions a state-space representation was derived for the symmetric (or longitudinal) motion in eq. (3.1) and asymmetric (or lateral) motion in eq. (3.3) taken from [39]. Where the values for A_{ij} and $\hat{A}_{ij}$ are non-dimensional and based on aircraft parameters, stability derivatives, moments of inertia and products of inertia. The values for B_{ij} and $\hat{B}_{ij}$ are non-dimensional control derivatives. In this research throttle control will be manually controlled and thus will not be part of the action space, but is included in the state space representation for coherency with the flight management system and autopilot interaction. The longitudinal states are defined by $\boldsymbol{s}^{lon}$ and the actions by $\boldsymbol{a}^{lon}$ in eq. (3.2). The lateral states are defined by $\boldsymbol{s}^{lat}$ and the actions by $\boldsymbol{a}^{lat}$ in eq. (3.4).

$$\begin{bmatrix} \dot{\hat{u}} \\ \dot{\alpha} \\ \dot{\theta} \\ \frac{\dot{q}\bar{c}}{V} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \begin{bmatrix} \hat{u} \\ \alpha \\ \theta \\ \frac{q\bar{c}}{V} \end{bmatrix} + \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \\ B_{41} & B_{42} \end{bmatrix} \begin{bmatrix} \delta_e \\ \delta_t \end{bmatrix} \tag{3.1}$$

$$\begin{aligned} \boldsymbol{s}^{lon} &= \begin{bmatrix} \hat{u} & \alpha & \theta & \frac{q\bar{c}}{V} \end{bmatrix}^T \\ \boldsymbol{a}^{lon} &= \begin{bmatrix} \delta_e & \delta_t \end{bmatrix}^T \end{aligned} \tag{3.2}$$

$$\begin{bmatrix} \dot{\beta} \\ \dot{\phi} \\ \frac{\dot{p}b}{2V} \\ \frac{\dot{r}b}{2V} \end{bmatrix} = \begin{bmatrix} \hat{A}_{11} & \hat{A}_{12} & \hat{A}_{13} & \hat{A}_{14} \\ \hat{A}_{21} & \hat{A}_{22} & \hat{A}_{23} & \hat{A}_{24} \\ \hat{A}_{31} & \hat{A}_{32} & \hat{A}_{33} & \hat{A}_{34} \\ \hat{A}_{41} & \hat{A}_{42} & \hat{A}_{43} & \hat{A}_{44} \end{bmatrix} \begin{bmatrix} \beta \\ \phi \\ \frac{pb}{2V} \\ \frac{rb}{2V} \end{bmatrix} + \begin{bmatrix} \hat{B}_{11} & \hat{B}_{12} \\ \hat{B}_{21} & \hat{B}_{22} \\ \hat{B}_{31} & \hat{B}_{32} \\ \hat{B}_{41} & \hat{B}_{42} \end{bmatrix} \begin{bmatrix} \delta_a \\ \delta_r \end{bmatrix} \tag{3.3}$$

$$\begin{aligned} \boldsymbol{s}^{lat} &= \begin{bmatrix} \beta & \phi & \frac{pb}{2V} & \frac{rb}{2V} \end{bmatrix}^T \\ \boldsymbol{a}^{lat} &= \begin{bmatrix} \delta_a & \delta_r \end{bmatrix}^T \end{aligned} \tag{3.4}$$

Characteristic dynamic motions of Cessna Citation I

The behavior of each eigenmode of an aircraft are different for each flight condition and each aircraft, to give an idea of the order of magnitude of a period of a few important eigenmodes, an overview of eigenmodes of the Cessna Ce500 'Citation' at $V = 60 m/s$ is given in table 3.1 taken from [39].

Mode	Target variable	Period [s]	Time to damp to half the amplitude [s]
Short Period	q	5.59	0.60
Phugoid	θ	32.13	81.00
Aperiodic Roll	p	-	0.22
Spiral	β, ϕ, ψ	-	6.78
Dutch Roll	p, r	2.00	2.34

Table 3.1: A collection of characteristic modes of Cessna Ce500 'Citation' to illustrate to the order of magnitude difference in dynamic behavior between the inner loop and outer loop variables

Adaptive flight control with reinforcement learning

A need for a adaptive autopilot was identified from the limitations of classical autopilots. To fulfill this need, a model-independent reinforcement learning technique is chosen to enable an adaptive autopilot for loss of control situations. The reinforcement learning controller will be embedded in the most inner loop where it controls angular body rates. Angular body rates have a direct dynamic relation with aircraft control surfaces, thus exhibits in the lowest learning complexity [24]. If a specific reinforcement learning technique permits, this reinforcement learning controller will be extended to the outer loop where it will control the Euler angles taking into account the fast and slow motions as elaborated in table 3.1. The controller built-up is similar to fig. 3.3.

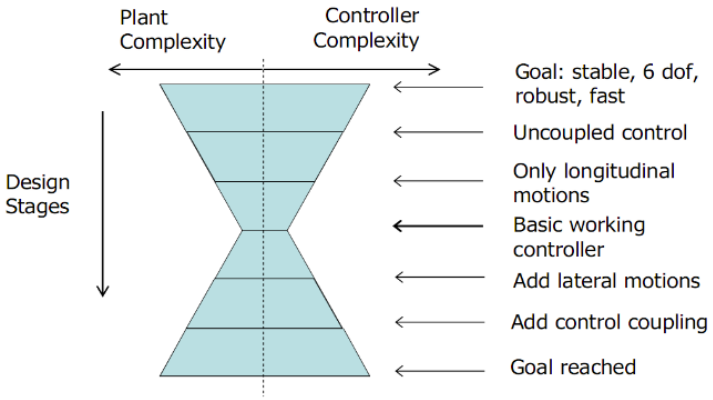

Figure 3.4: A schematic overview of design stages with regard to plant and controller complexity steps. Image is adapted from [67].

The six degrees of freedom motion of aircraft was uncoupled in longitudinal and lateral motion, this results in two state-space systems for each uncoupled motion given in eq. (3.1) and eq. (3.3) with their own state and action spaces eq. (3.2) and eq. (3.4). The first implementation of a reinforcement learning controller will only consider one of two uncoupled motions, in this case only the longitudinal motions for which a basic working controller will be designed. After which the plant complexity and controller complexity will be increased by adding the other uncoupled motion in this case, the lateral motions. The scheme shown in fig. 3.4 outlines the design phases as elaborated. **The initial scope of the implementation will be on basic working controllers for each of the uncoupled motions embedded in the inner loop, where a model-independent reinforcement learning technique for adaptive (fault-tolerant) flight control should perform control reconfiguration actions by performing online re-design of its control settings that derives from the current state of the plant and the received reward.**

3.2. Fundamentals of reinforcement learning

Most modern reinforcement learning techniques can be classified as a combination or extension of three fundamental classes of methods that solve for finite Markov Decision Processes (MDP), which is the mathematical basis of a reinforcement learning problem. Among these classes are Dynamic Programming (DP), Monte Carlo (MC) and Temporal Difference (TD) methods and these methods aim to solve the value function, each method having its drawbacks and advantages. In this section, the main elements of a reinforcement learning method and the aforementioned classes of methods are elaborated.

Markov decision processes

Markov Decision Processes (MDP) are a classical way to formalize sequential decision making in a mathematically idealized manner that applies to a reinforcement learning problem and for which precise theoretical statements can be made [61]. In other words, a MDP allows for framing the problem of learning from trial-and-error interactions to achieve a goal. The trail-and-error interaction can be conceptually described with an agent (learner and decision maker) that interacts with its environment (comprising everything outside the agent) as shown in fig. 3.5. The agent environment distinction represents the limit of the agent's absolute control, but not its knowledge. According to the agent environment interface, a trial is then executed by an agent taking an action based on state and error (same as negative reward) information from the environment. So the states are the basis for making the choices, the actions are the choices made by the agent, and the errors or rewards are the basis for evaluating these choices.

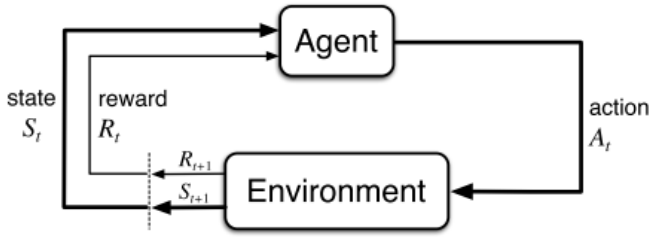

Figure 3.5: The Agent-Environment Interface describing the interaction between the agent and the environment for one time step. First the environment provides the agent with state and reward information. Then the agent performs an action based on received state and reward from the environment. The action from the agent interacts with the environment. The environment provides state and reward information for the next time step. Image is taken from [61].

For a reinforcement learning problem to be classified as, and be described by, the dynamics of a Markov Decision Process it needs to possess the **Markov Property**. A reinforcement learning problem adheres to the Markov Property when the probabilities given by p in eq. (3.5), completely characterize the environment's dynamics. The probability p of each possible value of S_t, R_t should only depend on the immediately preceding state and action, S_{t-1} and A_{t-1}. In addition, all information from the past agent-environment interaction that contribute to changes of future states should be included as state information. If these two conditions are met, a reinforcement learning problem is said to have the Markov Property.

$$p\left(s', r \mid s, a\right) \doteq \Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\} \tag{3.5}$$

Reward function

The utilization of a reward signal to encompass the idea of a goal is one of the most distinguishing features of reinforcement learning. By designing the reward signal a specific task for an agent can be set. A well-designed reward signal will take into account that when the agent maximizes the long term reward, then the agent has achieved the goal or task. So the rewards should indicate what the agent **what** needs to accomplish and not **how** the agent needs to accomplish it.

The reward can be computed from a discrete or continuous function. Here a continuous function is used to enable continuous control. The control task is to track a reference signal that is set by the pilot or autopilot, meaning that the task of the agent is to reduce the error between the measured signal and reference signal as given in eq. (3.6).

$$e(t) = x(t) - x_d(t) \tag{3.6}$$

The choice of reward function largely influences how fast the agent learns and what it learns, so varying the reward function largely influences the behavior of the agent. In this research the focus is mainly on the techniques used for the agent and not its environmental influences in this case the reward function. Thus a reward function that is best suited for this research is taken from literature.

A formulation is developed by [28] that gives both feedback and feedforward parts of the control input simultaneously and thus enables RL algorithms to solve the tracking problems without requiring complete knowledge of the system dynamics The formulation is an adaptation of the Linear Quadratic Regulator Problem. For this research a simplified version of this formulation is used as given in eq. (3.7) which is also used in [25] and section 5.2.2.

$$r_{t+1} = r\left(s_t^R, s_{t+1}\right) = -\left[\boldsymbol{P}s_{t+1} - s_t^R\right]^T Q\left[\boldsymbol{P}s_{t+1} - s_t^R\right] \tag{3.7}$$

The LQR reward function leaves room for tuning the P and Q matrices, which depend on the task at hand, this and amongst other forms of altering the reward function is called reward shaping.

Reinforcement learning methods comprises two elemental functions that make use of the dynamics of a MDP process. The two elemental functions encode the goal of reinforcement learning methods. The goal being the search for a policy that achieves a lot of reward over the long run. The two elemental function are called the policy function and the value function.

Policy function

An agent selects its actions based on states that are received from its environment. The agent translates a state to an action by using a map of states that correspond to a probability of selecting each possible action, this is known as a policy. Thus a policy is a probabilistic rule by which the agent selects actions as a function of states. The policy function $\pi(a \mid s)$ defines a probability distribution over $a \in \mathscr{A}(s)$ for each $s \in \mathscr{S}$. The probabilistic rule is updated through a value function. For a learned value function, it is said that the policy is improved due its experience over time. A improved policy is created by selecting the action that maximizes the value function of the original policy as is done in eq. (3.8), this is the procedure of **policy improvement**.

$$\pi'(s) = \arg\max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right] \tag{3.8}$$

The approach of action selection used for policy improvement is called **greedy**, the term greedy describes a decision procedure that selects actions based on only shortsighted considerations. Greedy approaches do not consider the possibility that these shortsighted considerations might inhibit future access to better actions.

Value function

The value function can be computed for a state-value function and a state-action value function. The state-action value function is predominantly used for Q-learning methods [70] [71] due to the focus on policy gradient methods, the Q-learning methods are not included in this literature review. Thus for sake of simplicity only the predictions of the state-value functions are given, but can be easily extended to state-action value functions.

The agent's goal is to maximize the total amount of received reward. Mathematically speaking, this equals the maximization of the expected value of the cumulative sum of a received reward signal (a scalar). This is mathematically expressed in a value function given in eq. (3.9). Where the value function evaluates future rewards that can be obtained by following a specific or random action resulting from a policy function. Depending on the problem setting the value function can prioritize maximizing immediate or delayed reward by setting this behavior with the discount rate γ ranging from 0 to 1. A value function for following a policy is given in eq. (3.9) where the value function is equal to the expected return when following a policy. The expected return depends on future discounted rewards given states at the current time step from the environment.

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{ for all } s \in \mathscr{S} \tag{3.9}$$

A challenge in reinforcement learning has been to have a reliable and practical prediction of the expected return. Different strategies can be applied to determine the value of future rewards or expected return depending on the use-case and desired accuracy. In general, it can be said that the variety of reinforcement

learning techniques are centering around finding appropriate methods that can best approximate optimality in predicting expected return given specific design requirements.

A fundamental property of value functions used throughout reinforcement learning and dynamic programming, for any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states given by the Bellman equation eq. (3.10)

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right], \quad \text{for all } s \in \mathcal{S} \tag{3.10}$$

For each a, s' and r, the probability is computed, $\pi(a \mid s)\sum_{s',r} p\left(s', r \mid s, a\right)$, the quantity in brackets is weighted by that probability, then all possibilities are summed to get an expected value. As a result, the value function v_π is the unique solution to its Bellman equation. The Bellman equation forms the basis of a variety of ways to compute, approximate, and learn v_π through the Generalized Policy Iteration.

Generalized Policy Iteration and Approximation of Optimality

A general concept for most reinforcement learning methods that encompasses the process of finding an optimal policy and optimal value function is the Generalized Policy Iteration (GPI). GPI emphasizes the common denominator of the variety of policy iteration methods, which is that they globally have consecutive interactions between policy evaluation and policy improvement that leads to an approximation of an optimal value function and optimal policy as depicted in fig. 3.6. The differences in policy iteration methods lies in the amount of iterations taken within the policy evaluation and the timing of updating the values related to policy evaluation and improvement.

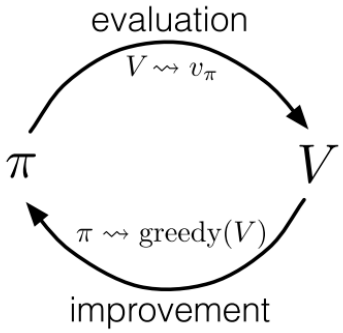

Figure 3.6: Local interaction between policy evaluation and policy improvement for the Generalized Policy Iteration [61].

A GPI starts with a policy evaluation of a initial policy, this is followed by a policy improvement. If a policy improvement has been executed that resulted in a new policy $\pi'(s)$, then a new value function $v_{\pi'}(s)$ can be computed, that evaluates the policy, this is known as **policy evaluation**. The new value function will be more accurate than the original value function $v_\pi(s)$ and from the new value function a new policy improvement $\pi''(s)$ can take place that results in an even better policy. This procedure of consecutively executing policy evaluation and policy improvement will eventually result in finding an optimal policy. This procedure of finding an optimal policy is known as **policy iteration** and is visualized in fig. 3.7.

The policy iteration continues until the current policy is consistent with a value function that has stabilized, and when the policy is greedy with respect to the current value function (eq. (3.10)). In other words, if a policy has been found that is greedy with respect to its own evaluation function, then the policy iteration is stabilized. As a result of this greedy policy and starting with policy evaluation, is that the value function given by the Bellman equation will equal the Bellman optimality equation eq. (3.11). Thus, the optimal value function $v_*(s)$ is found as first.

$$v_*(s) = \max_a \sum_{s'r} p\left(s', r \mid s, a\right)\left[r + \gamma v_*\left(s'\right)\right] \tag{3.11}$$

The advantage of starting with optimizing the value function is that it allows for organizing and structuring the search for optimal policies by enabling the consistency condition of the Bellman equation to apply indirectly for the policy function. All in all, meaning that the policy and value function are optimal for reinforcement learning methods that use policy iteration procedures that include policy evaluation and policy improvement.

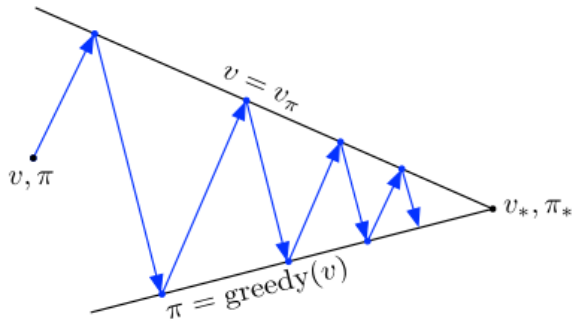

Figure 3.7: Global interaction between policy evaluation and policy improvement, resulting in a monotonic improvement of policy and value function visualized as a iterative process. Where the blue line depicts the error and the blue dots on each line a iteration of its respective function [61].

In approximating optimal behavior, there may be many states that the agent faces with such a low probability that selecting suboptimal actions for them has little impact on the amount of reward the agent receives. The online nature of GPI in reinforcement learning makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states. This is one key property that distinguishes reinforcement learning from other approaches to approximately solving MDPs.

Finding an optimal policy through GPI, is dictated by its value function. In this regard three main reinforcement learning methods can be clearly and distinctively identified by their method of computing, approximating or learning the value function. These methods being Dynamic Programming (requires complete and accurate model), Monte Carlo (not well suited for step-by-step incremental computation), and temporal difference learning (require no model, fully incremental, but more complex) these methods differ in efficiency and speed of convergence. The relationship between these methods is a recurring theme in the theory of reinforcement learning. In addition, these ideas and methods blend into each other and can be combined in many ways. The TD(λ) algorithm, seamlessly unifies TD with MC.

In some cases, GPI can be proved to converge, most notably for the classical DP methods that we have presented in this chapter. In other cases convergence has not been proved, but still the idea of GPI improves our understanding of the methods

Dynamic Programming

Dynamic Programming (DP) is a family of algorithms that form an fundamental foundation for the understanding of reinforcement learning methods throughout literature. Thus aspects of DP methods already have been elaborated in the form of the value function and the Generalized Policy Iteration.

DP methods can **compute** optimal policies when given a perfect model of the environment. Computation of the optimal policy is enabled through recursive relationships expressed in the Bellman equation in eq. (3.10).Where the Bellman equations are turned into step-by-step update rules that improve the approximation of the value function, given in eq. (3.10). The solution to Bellman equations can be seen as a system of equations with each state having its own equation. So if there a n states, then there are n equations in n unknowns. Then taking the Bellman optimality equation eq. (3.11) and given that the dynamics of the environment are known, the nonlinear system of equations is solved and a optimal value function $v_*(s)$ is obtained. Though this approach is straightforward, it is tedious and requires great computational effort. An iterative approach to this policy evaluation process reduces the computational effort. Then a optimal policy is found with less computational effort using the iterative policy evaluation in the Generalized Policy Iteration.

This approach of solving the Bellman optimality equation is seldom directly useful. As it involves an exhaustive search where it is required to look ahead at all possibilities, followed by computing the probabilities of occurrence and the expected returns. In addition, solutions for Dynamic Programming methods rely on at least three assumptions that are seldom true in practice: firstly, the dynamics of the environment need to be known accurately; secondly, enough computational resources are required to complete the computation of the solution; and thirdly, the Markov property needs to be adhered to.

Monte Carlo, Temporal Difference learning and Policy Gradient methods can be viewed as attempts to

achieve much the same effect as DP, namely solving the Bellman optimality equation, only with less computation and without assuming a perfect model of the environment.

Monte Carlo Methods

Monte Carlo (MC) methods are estimation methods that provide ways of solving the reinforcement learning problem that involve averaging over many random samples of actual or simulated returns. Whereas the value functions are computed step-by-step from knowledge of the MDP dynamics with DP methods. In MC methods the value functions are learned from complete sample returns with the MDP. In other words, these methods do not assume knowledge of the environment's dynamics. They only require interactions with an environment from which sample sequences of states, actions, and rewards are collected, this process of collecting sample sequences is called **experience**.

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)] \tag{3.12}$$

The only requirement for the environment model is the ability to generate sample transitions, thus not requiring the complete set of probability distributions of all possible transitions to be known. The latter being the case for DP methods. In addition, MC methods update their value function, given in eq. (3.12), on a episode-by-episode to compute their return G_t and not on a step-by-step basis as is the case with DP methods. As a result, MC methods do not bootstrap, meaning their value estimates are not updated on the basis of other value estimates. This may lead to smaller consequences when the Markov property is violated. The GPI also applies to MC methods to find the optimal value function and policy function, though finding an optimal policy that maintains sufficient exploration is more challenging for MC control methods when using action-value function for policy evaluation.

Temporal Difference Learning

Temporal Difference (TD) learning is one of the three fields that make up the field of reinforcement learning. TD learning interrelated the field of optimal control with psychology of animal learning, where the key concept is the temporal difference errors δ_t that are used to estimate an increment in predicted value [61]. For this increment the TD method takes the difference between the one time step ahead return $R_{t+1} + \gamma V(S_t)$ and the current state value $V(S_t)$. Though this is not possible, as the agent is learning from experience. So the future return is estimated using the current reward plus the current value estimate and then discounted with γ. The temporal difference is then calculated by subtracting the future return with the previous value estimate. The value function update as described is summarized in eq. (3.13).

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] = V(S_t) + \alpha \delta_t \tag{3.13}$$

The TD method is similar to the Monte Carlo methods as it learns its value function directly from raw experience without a model of the environment's dynamics. In another aspect the TD method is similar to Dynamic Programming as it bases its updates of the value function using previous estimates – using bootstrapping. The TD method combines the best of both MC and DP methods, thus enabling the method to be model-independent by learning through generated experience through interaction with the environment and the method can be applied online as it can update its value and policy function by just waiting one time step.

Model-independent and model-dependent

Reinforcement learning problems can be categorized in multiple ways, one of them is to categorize them as either model-dependent or model-independent methods. In model-dependent methods the agent requires information about the transition probabilities of the environment to predict the value for the next time steps. The information about the transition probabilities of the environment can be referred as model information or dynamics of the environment, this information is either provided or learned by the agent.

In general agents that use an learned model are more sample efficient than model-independent agents. Model-independent agents can be less sample efficient as they have less information about future values and thus needs more experience to make better decisions. Model-dependent methods can still result in sample efficient learning if the model is too complex and results in sample inefficient learning of the model. A rule of thumb that can be used to decide whether to choose for a model-dependent or model-independent method. In case that the model is easy to learn then a model-dependent method provides a good baseline, if the

policy is easy to learn then a model-independent method can be a better choice. On a last note, the model-dependent agents are more reliant on accurate models of the environment, as indicated in chapter 1 thus model-independent methods are desired for flight control.

3.3. Policy gradient reinforcement learning

Policy gradient reinforcement learning (PGRL) are methods that can learn a parameterized policy that can select actions without using a value function. PGRL does this by employing gradients to update a policy parameter vector on each time step $\boldsymbol{\theta_t}$ in the direction of an estimate of the performance gradient ∇J with respect to the policy parameter that has a step-size α as defined in eq. (3.14). In contrast to Q-learning methods, policy gradient methods do not have to rely on value functions. If they do use value functions than they largely rely on state-action value functions. Policy gradient methods used in this study learn a parameterized policy $\pi(a|s,\boldsymbol{\theta})$ where action selection can be performed without utilizing action-value functions. Though an action-value function may still be used to learn policy parameters in order to increase performance of more complex problems. For J a stochastic gradient ascent method is applied to maximize the performance. Hence the name policy gradient reinforcement learning. If a reinforcement learning algorithm adheres to eq. (3.14), then it is a policy gradient method.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\nabla J}(\boldsymbol{\theta}_t) \tag{3.14}$$

For policy gradient methods to be compatible with existing function approximation methods a challenge had to be solved. The performance function depends on both action selection and distribution of states corresponding to that action selection. Both of these are on their turn affected by the policy parameter. The Policy Gradient Theorem in eq. (3.15) addresses the unknown effect of policy parameter changes on the state distribution by excluding the derivative of the state distribution [72] and instead using a steady-state distribution under a policy $\mu(s)$ or also known as the on-policy state distribution under π. The Policy Gradient Theorem provides an analytical expression for the performance gradient with respect to the policy parameter in eq. (3.14) and allows for applying stochastic gradient ascent methods to estimate the performance gradient. Instead of applying a first-order gradient method, a second-order policy gradient method called natural policy gradient [27] could also be used and classify as policy gradient method. The natural policy gradient requires less iterations for convergence but needs much more computation per iteration and adds more complexity, so the focus of this study is mainly on the first-order methods.

$$\nabla J(\boldsymbol{\theta}) = \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla\pi(a \mid s,\boldsymbol{\theta}) \tag{3.15}$$

In regard to policy gradient reinforcement learning methods, the performance gradient is always an approximation – here by definition – as policy gradient reinforcement learning methods are an approximate way of solving reinforcement learning problems. So throughout this report when $\nabla J(\boldsymbol{\theta})$ is used to indicate the performance gradient, then the correct notation should be $\widehat{\nabla J(\boldsymbol{\theta}_t)}$ but to improve readability the approximation sign will be omitted.

An advantage of parameterizing policies using parameterization such as numerical preference is that it will lead to a natural way of learning optimal stochastic policies. As actions are selected using numerical preference which are estimated probability distributions that are based on continual dependence of their previous estimations, whereas action-value methods have no natural way of finding stochastic optimal policies. As a consequence of using such parameterized policies is that in the limit the stochastic policy can approach a deterministic policy. In addition, when its chosen for a continuous policy parameterization then the action probabilities will change smoothly as a function of the learned parameter. Together with the continuous policy parameterization and the policy gradient theorem, this can provide for strong convergence guarantees, continuous function approximation and continuous action control when using policy-gradient methods.

Though there are inherent disadvantages to policy gradient methods, as they have poor sample efficiency and suffer from high variance [61]. A way to tackle high variance is to increase batch size as to reduce variance. But that in its turn will result in a lower sample efficiency, as it requires more samples. At this point a trade-off should be made between low variance or high sample efficiency. In addition, there is no mathematical assurance of converging to a globally optimal solution, thus having the risk of only converging towards local optima ([72]). So a search for methods that aim to solve for these disadvantages while using policy gradient methods was initiated and resulted a range of methods that will be presented in section 3.3.1, section 3.3.2 and chapter 4.

3.3.1. REINFORCE

REINFORCE is a policy gradient method introduced by Williams [72] can be seen as the simplest policy gradient technique since it directly follows from the Policy Gradient Theorem. Its acronym stands for "REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility", which is describes the form of this algorithm in eq. (3.16).

$$\Delta w_{ij} = \alpha_{ij}\left(r - b_{ij}\right) e_{ij} \tag{3.16}$$

Where Δw_{ij} is the reward increment and nonnegative factor is α_{ij} these corresponds to the second term in eq. (3.14) and α in eq. (3.14), respectively. In order to go from eq. (3.15) to the expression in eq. (3.16), then eq. (3.15) needs to be sampled using Monte Carlo. Sampling over the expectation and rewriting results in the offset reinforcement $\left(r - b_{ij}\right)$ which is the full return minus a baseline function and the Characteristic Eligibility e_{ij} which is the gradient of the policy divided by the policy function.

The expected update of a policy gradient is equal to the gradient of expected reward, thus that this algorithm is an instance of stochastic gradient ascent. This assures that the algorithm has robust convergence properties. REINFORCE is using a Monte-Carlo method, which means it updates in retrospect of a finished episode, thus executes one policy iteration after each episode. Monte-Carlo methods are generally of high variance and thus produce and result in slow learning, but are unbiased in their estimates. A baseline function can be used or even a random variable as long as it does vary with the policy parameter. A baseline function that can greatly speed up learning by reducing the variance of the policy gradient is the state-dependent value function while still remaining unbiased. This due to the fact that the change in state value function with respect to the policy parameter is equal to zero.

Though finding a good baseline function in general and in the form of a state value function is another challenge. So learning the baseline provides a solution, but how the baseline is learned determines if the policy function will get biased. In the Actor-Critic the value function is learned through bootstrapping by propagating the temporal difference error as a feedback signal for the Actor network.

3.3.2. Actor-Critic Design

A learned policy is equal to a control law and determines how an agent should act. This is the Actor component of Actor-Critic methods. To give this agent extra guidance, it uses a learned state-value function that is used to estimate if the executed action has added more or less value by providing the temporal difference error δ_t (defined in eq. (3.13) to update the Actor, this component is called the Critic. The update rule for the policy parameter of the Actor-Critic is provided in eq. (3.18). The parameters within the Actor and Critic functions are often estimated using neural networks, making these functions suitable for differentiation, back propagation, and high-dimensional state space and continuous action space [62]. The neural network of the Actor can learn a probability distribution by estimating a mean μ and standard deviation σ of a Gaussian distribution by using the policy parameterization in eq. (3.17), from which continuous actions are sampled [14].

$$\pi(a \mid s, \boldsymbol{\theta}) \doteq \frac{1}{\sigma(s,\boldsymbol{\theta})\sqrt{2\pi}} \exp\left(-\frac{(a-\mu(s,\boldsymbol{\theta}))^2}{2\sigma(s,\boldsymbol{\theta})^2}\right) \tag{3.17}$$

Although the REINFORCE-with-baseline in section 3.3.1 learns both a policy and a state-value function, it is not considered an Actor-Critic method since its state-value function is only used as a baseline and not as a Critic. The Critic makes use of bootstrapping as performed in one-step temporal difference methods. The difference between an Actor-Critic method and REINFORCE lies in the state-value estimate, with REINFORCE it does not estimate its predicted state value using bootstrapping, thus making REINFORCE an unbiased method. Bootstrapping for state-value function estimation introduces bias within the Actor-Critic agent but is a necessary evil for the same reason that bootstrapping Temporal Difference methods are often superior to Monte Carlo methods, as they substantially reduce variance.

$$\nabla J(\boldsymbol{\theta}) = \delta_t \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)} \tag{3.18}$$

A preference for methods with an Actor-Critic Design is identified on its advantages that suit the continuous problem domain and real-life applications. The advantages are that these methods have an analytical basis, typically learn faster, and interfaces well with existing back-propagation techniques [72]. In addition, when using a stochastic policy parameterization, then an Actor-Critic agent is able to explore the action space

and have a natural good exploration-exploitation trade-off. The Actor-Critic Design is used as a baseline for continuous and model-independent control but requires a higher sample efficiency and extension to large continuous domains to make it more suitable for real-life applications. The search for methods to increase sample efficiency and scalability of the Actor-Critic Design is set forth in a literature study of the state-of-the-art in chapter 4.

3.4. Conclusion

The goal of this chapter is to provide an answer to the first research question: The aim of this chapter is to provide a theoretical foundation of this research by answering the sub-questions of the first research question in section 2.2 and restated here: **What the are requirements for a fixed-wing aircraft flight controller?**. And by elaborating on the fundamentals of reinforcement learning and policy gradient reinforcement learning.

In answering the sub-questions of the first research question, it is found that from design principles of automatic flight control systems that the focus will be on developing a basic model-independent reinforcement learning controller for the uncoupled motions in the inner control loop. Here, attention should be paid to the timescales involved in each control loop and thus the controlled state. The uncoupled motions are symmetric and asymmetric motions that respectively need a controller in the longitudinal and lateral directions. The model-independent reinforcement learning method should perform control reconfiguration actions that re-design its control settings in an online fashion by using information about the current state of the environment and the received reward signal.

4

Literature study part II: State of the Art

In this chapter, the state-of-the-art non-hierarchical and hierarchical policy gradient reinforcement learning methods are presented that built on the fundamentals presented in chapter 3. The methods here are reviewed with regard to the flight control requirements that were reviewed in the aforementioned chapter, and aims to answer two sub-research questions that can also be found section 2.2 and restated here:

2. **What are the current state-of-the-art reinforcement learning methods that are suitable for flight control?**

   (a) *What is the most suitable state-of-the-art policy gradient reinforcement learning technique?*

   (b) *What is the most suitable state-of-the-art hierarchical policy gradient reinforcement learning technique?*

In answering the stated research questions, a focus on model-independent Actor-Critic Design is taken, and in this regard, two fields were identified and elaborated; one being Actor-Critic Design in Deep Reinforcement Learning in section 4.1, and the other being Actor-Critic Design in Approximate Dynamic Programming in section 4.2. In addition to Actor-Critic Design section 4.3, policy gradient optimization methods have seen wide adoption in Deep Reinforcement Learning and can be added on top of Actor-Critic Design as will be elaborated in section 4.3. Another field of methods that are known for providing possible performance improvements on existing reinforcement learning methods are Hierarchical Reinforcement Learning methods through Hierarchical Design, presented in section 4.4. This chapter is concluded in section 4.5 which will answer the stated research question.

4.1. Actor-Critic Design in Deep Reinforcement Learning

The field of Deep Reinforcement Learning (DRL) is known for its end-to-end learning capabilities by extending traditional reinforcement learning methods with non-linear function approximators to estimate large and complex value and policy functions. In DRL, often use is made of Deep Neural Networks (DNN) as the non-linear function approximator. By extending RL with DNN makes the conventional RL methods more capable of handling complex environments with high-dimensional and continuous state-action spaces, though this is not straightforward, especially for policy gradient methods. This sets out two frameworks found in DRL that focus on policy gradient reinforcement learning with Actor-Critic Design. In section 4.1.1 the Deep Deterministic Policy Gradient method is elaborated and focuses sample efficiency. In section 4.1.2 the Asynchronous and Synchronous Advantage Actor-Critic framework for stabilized learning is presented where also a Generalized Advantage Estimation (GAE) function was implemented with an Actor-Critic design.

4.1.1. Deep Deterministic Policy Gradient

The inception of Deep Deterministic Policy Gradient (DDPG) methods was through identification of the advantages and disadvantages of Deep Q-Learning methods such as Deep Q-Networks (DQN) [36]. DQN's are one of the first Q-Learning [71] methods that could harness the advantages of deep neural networks to learn control policies directly from high-dimensional sensory data. In the past, large and deep neural networks in

combination with reinforcement learning often resulted in unstable learning. In addition to unstable learning, to this day, there is no common theoretical framework from which the theoretical performance can be guaranteed. Thus in the past, the use of large neural network architectures for learning a value or action-value function was avoided.

The advantages of DQN are clear, though it was found that they are not applicable to physical control tasks, as these tasks require controllers to handle continuous and possibly high-dimensional action spaces, whereas Q-Learning methods can only handle discrete and low-dimensional action spaces. The disadvantages and advantages of DQN inspired the development of Deep Deterministic Policy Gradient [33] that uses the Deterministic Policy Gradient (DPG) [56] algorithm at its foundation while implementing techniques from DQN to stabilize learning with large neural networks. The two novelties that enabled stabilized learning with deep neural networks for DQN was the use of a replay buffer [3] and a separate target network for calculating the return.

The focus of DPG [56] lies in solving the sample inefficiency encountered in policy gradient reinforcement learning applied to high-dimensional action spaces. In general a basic policy gradient method is by design employing a stochastic policy function. Stochastic policy gradient method are in general less sample efficient than deterministic policies, as these stochastic gradients may require more samples as it does not only integrate over state space, but also over the action space. The aim of the authors that introduced DPG was to have a more sample efficient and stable learning method for high-dimensional tasks while retaining the exploration behavior of stochastic policies. The answers to this was a Deterministic Policy Gradient that employs a target policy behavior that learns a deterministic policy from an exploratory behavior policy. This was enabled by utilizing their main finding that a deterministic policy gradient is in the limit of a stochastic policy gradient as the variance goes to zero. Though this combination of having the best of both worlds comes at a cost in regards to its applicability for the context of this research. The DPG method only has the beneficial characteristics as described in an off-policy setting. Off-policy methods are characterized by how the agent updates their current probability distribution by using a different probability distribution and often requires multiple samples per policy gradient update.

In flight control, the distribution of unforeseen circumstances during flight is not known and can change abruptly. An off-policy RL method may not be sample efficient and exploratory enough to adjust its policy while having to stabilize the aircraft during operation. Thus off-policy RL methods are less applicable for adaptive flight control during online operation. Though the concept of using a target network to stabilize learning is insightful for other applications and will be applied to flight control as seen in section 4.2.2.

4.1.2. Synchronous and Asynchronous Advantage Actor-Critic

Asynchronous Advantage Actor-Critic (A3C) is an Actor-Critic method using a simple and lightweight framework that trains deep neural network controllers by asynchronous gradient descent [37]. The asynchronous method employs parallel actor-learners in value- and policy-based methods that are optimized through asynchronous gradient descent. Employing this asynchronous framework, the authors [37] aims at reducing non-stationarity and decorrelating the sequence of encountered data by online RL algorithms. The authors mainly study their framework with deep neural networks that share parameters in combination with parallel actor-learners. There it was found that these actor-learners have a stabilizing effect on the training of deep neural network controllers and increase data efficiency during training. A3C was able to execute continuous motor control tasks as defined in MuJoCo [64].

For practical applications where low training time expressed as wall-clock time – data efficiency – is important, then A3C would be a good candidate. For flight control, it is important to have a high sample efficiency and has more priority than a high data efficiency. The use of parallel actor-learners in A3C can be implemented in the fine-tuning phase during the detailed design phase of a flight controller.

A deterministic and synchronous variant of the A3C was proposed by researchers of OpenAI that yields the name Advantage Actor-Critic (A2C) [73]. A2C uses a coordinator that is waiting for all Actors to finish with their iteration and then updates the global parameters altogether. In the consecutive iteration, the agents start with the same policy. By updating the global parameters altogether using the synchronized gradient update, the agents are trained in a cohesive fashion, whereas A3C used an asynchronous gradient update. The cohesive learning of A2C might explain the cases that the agent requires less samples than A3C to learn a good policy.

In [50] the author found that A2C is learning faster than A3C. Additionally, the author replaced the advantage function that was used by [37]. Instead, a Generalized Advantage Estimation (GAE) function [49] was used with the A2C that resulted in even faster learning, as the GAE is a baseline function for policy gradient

methods that can reduce the variance during learning. Additionally, the GAE, when implemented in an Actor-Critic method provides for tuning of the variance and bias by employing techniques seen in TD(λ)-learning. When using a GAE function as the baseline for policy gradient methods and introducing a small amount of bias, then the variance can be greatly reduced. As a result, the learning is faster during the initial phase of training, making it more sample efficient than methods not using GAE [46].

4.2. Actor-Critic in Approximate Dynamic Programming

Traditionally Dynamic Programming (DP) methods may be considered not practical for very large problems [61]. The field of Approximate Dynamic Programming (ADP) stems from Dynamic Programming (for more historical context, see chapter 1). In ADP, the focus is on applying DP methods to large problems that are continuous and possibly nonlinear by tackling the curse of dimensionality. Limitations concerning the problem size of DP methods are alleviated by using a neural network for nonlinear function approximation of value functions and/or policy function, though not limited to neural networks. ADP methods can have critic-only variants, in section 4.2.1 focus will only be on the Actor-Critic variant that are the Adaptive-Critic Design methods. Recent developments made it possible to have ACD's that have online-only training phase this method is called Incremental Dual Heuristic Programming (IDHP) method and presented in section 4.2.2.

4.2.1. Adaptive-Critic Design

The Adaptive-Critic Design (ACD) methods inherit the advantages of Dynamic Programming and tackle the disadvantages with a Critic (and sometimes Actor) setup and approximating the value function and, if applicable, policy function with non-linear function approximators such as neural networks that are updated using the temporal difference error. The advantages of DP being bootstrapping and their potential to solve optimal control problems in a principled and mathematical way. ACD methods solve the backward direction of the 'smart' exhaustive search performed by DP methods that limited DP methods to be applicable for real-time control [45]. For ACD, the search is redirected into a forward search by employing a critic network with non-linear function approximation that reduces the dimensions, and computational expense of the analytical solution of the Bellman's equation [44].

In general, the Adaptive-Critic Design [45] algorithms, as the name suggests, are centered around alterations of the Critic network that vary in their input, output, or having an Actor network. The three main variations of ACD's can be derived from one of its variation being the Global Dual Heuristic Programming (GDHP). The other two variations are scaled-down versions of GDHP (the more advanced ACD), which are Dual Heuristic Programming (DHP) and Heuristic Dynamic Programming (HDP). The HDP method is the most simplest ACD form. One of the differences between these methods lies in the Critic output of the value function (corresponding to HDP), either being the value function (corresponding to DHP), the derivative of the value function, or both (corresponding to GDHP). GDHP, , and HDP have Critics that employ a non-linear function approximation for estimation of a state-value function. The aforementioned methods also have a variation on their Critics which approximates the state action-dependent value function these methods are then prefixed with AD- and stands for Action-Dependent, which means that the Critic is estimating an action-state value function by using the output of an Actor network that is only connected to the Critic. A complete overview is given in table 4.1, where it is also summarized if either the Critic, Actor, or both need a state-transition model during training.

Table 4.1: Overview of the different components that makes up the variety of ACD's, inspired by [24].

ACD	Critic topology		State-transition model requirement	
	Input	Output	Actor	Critic
HDP	s	v	yes	no
ADHDP	$[s \quad a]$	q	no	no
DHP	s	$\frac{\partial v}{\partial s}$	yes	yes
ADDHP	$\begin{bmatrix} s & a \end{bmatrix}$	$\begin{bmatrix} \frac{\partial q}{\partial s} & \frac{\partial q}{\partial a} \end{bmatrix}$	no	yes
GDHP	s	$\begin{bmatrix} v\frac{\partial v}{\partial s} \end{bmatrix}$	yes	yes
ADGDHP	$\begin{bmatrix} s & a \end{bmatrix}$	$\begin{bmatrix} q & \frac{\partial q}{\partial s} & \frac{\partial q}{\partial a} \end{bmatrix}$	no	yes

An interesting note is that the simplest ACD with its Action-Dependent variant, Action-Dependent Heurist

Dynamic Programming (ADHDP), is equal to the Q-Learning [71] method adjusted for the continuous domain and without non-linear function approximation. Like HDP and ADHDP, Q-learning is also not requiring the state-transition probabilities of the environment to be known thus HDP and ADHDP are model-independent methods. Unfortunately, the ADHDP and HDP and their Critic networks have the potential in being too coarse. In addition, the DHP and GDHP outperform HDP in success rate and tracking precision [68]. As a consequence, these methods are often not considered for (large) continuous control problems.

DHP and GDHP fall in the category of Actor-Critic methods known to Deep Reinforcement Learning. Their Action-Dependent variants are disregarded here as these are Critic-only methods and thus fall out of the scope of this research. Continuing to DHP and GDHP, their main advantage over HDP is in the direct estimation of the derivatives of their optimization criterion (cost-to-go function) that is contained in the derivative of the value function. This is instead of estimating the optimization criterion itself. The information provided by these gradients was identified as the key success factor in finding better policies than HDP methods by [41] and [45]. GDHP differs from DHP in its Critic architecture instead of using just one Critic network as for DHP and HDP, GDHP employs parallel Critic networks that estimate the value function and the derivative of the value function separately. The marginal performance gain that comes with additional computational complexity results in a considerably higher computational cost for GDHP when compared to DHP. This was considered not a satisfactory trade-off for practical applications [45]. The ability to control real-time continuous problems through ACD's still requires a state-transition model of the environment, that is either learned or provided, for the most powerful methods. The state-transition model is required during training of the Critic or Actor or both. A summary for each ACD method is provided in table 4.1. In the following section, section 4.2.2, the review will mainly focus on the solution to tackle the model-dependent characteristic of ADP and DHP methods.

4.2.2. Incremental Approximate Dynamic Programming

Traditional ACD's presented in section 4.2.1 rely on a representative system model that is used during offline training in order to be the most effective during the online training phase and online operation of a continuous system. Incremental Approximate Dynamic Programming (IADP) methods take away the reliance on a representative system model and the offline training phase. The main advantage of IADP methods such as Incremental Dual Heuristic Programming (IDHP) [80] is in the insight, obtained from adaptive controllers such as Incremental Nonlinear Dynamic Inversion (INDI) methods [52] [58] [35], that model-dependent methods can be turned into model-independent methods using online system identification methods such as Recursive Least Squares (RLS) and estimating an incremental model of the environment. This alleviates the need for an accurate and global environment model by identifying locally the dynamics of the environment during operation by using only two samples, though the incremental model requires a sufficiently high data sampling frequency for its estimation. IADP methods are considered model-independent as the agent does not need any a priori information of the environment's dynamics during training nor online identification of the global nonlinear environment model, but only the online identified incremental environment model [79].

The opportunity that was identified to use incremental and online identification methods is more of an alteration to the plant model than in the agent itself. The agents are still using state-transition models of the environment – though identified online, incremental and local – to train the Actor and Critic networks of a DHP method. Though the IDHP method in [80] provides adjustments in the Actor and Critic update rules to facilitate a simpler learning model and to be able to interface with an incremental and online model estimation of an environment. The IDHP method is seen as an extension Incremental Heuristic Dynamic Programming (IHDP) [79] [78], but is excluded in this review as the Actor network is not connected to the environment model. In addition, HDP was seen as inferior to DHP as was explained in section 4.2.1.

The application of IADP methods – and especially IDHP – to aircraft models have been studied. A comparison between the IDHP and DHP methods for flight control indicates that IDHP methods accelerate online learning, improve precision in reference signal tracking, and handle a wider range of initial conditions or states [80]. IDHP has also proven to successfully provide for adaptive control in new and unstable systems, while DHP fails to, thus it was concluded that IDHP was better equipped for fault-tolerant control. Though this was mainly studied for a simplified flight model. In another study, the author [25] applied the IDHP method on a more complex flight model with a high-dimensional state space to enable full-body rate control, while retaining the aforementioned advantages of IDHP. This was facilitated through a target Critic network that stabilized learning for larger state spaces, this adaption of target networks can also be seen in DQN's [36].

The knowledge gained from research on cascaded Actor networks applied in flight control with ADP meth-

ods [18] and ACD methods [65] was combined with the success of IDHP to enable full altitude control with measurement noise and atmospheric gust [30]. The results from this research indicate that the introduced measurement noise added to overall learning stability as it repressed aggressive policy, whereas the cascaded Actor network was not able to handle the higher dimensional state space as a consequence of incorporating atmospheric gust in the flight model.

The lessons learned from applications of ADP and IADP methods in flight control; variations of Critic networks can facilitate stabilized learning for higher dimensional state spaces, cascaded Actor networks have the potential to provide full longitudinal flight control, online identification of incremental models contribute to shorter online learning for model-dependent agents, and vary in experiment setup in regards of kind of controller and testing. The testing is often performed in regards of control precision, adaptability, and robustness to specifically added model complexities.

Although the ACD and IADP methods are suitable for adaptive flight control, the methods introduced here learn deterministic policies. Deterministic policies have as a drawback that it needs to incorporate artificial noise or an exploration signal for learning good exploratory policies [15]. This may require careful analysis of the noise signals and should be modeled to resemble actual noise from sensors, actuators, and turbulence. As a result, the performance of these methods relies on satisfactory exploration of the state space by exploration signals that should guarantee the convergence of learned policies. A desire for a method that by itself learns a good exploratory policy without persistent excitation is identified here.

4.3. Policy Optimization algorithms

Policy optimization algorithms, in this review, alternates between sampling and optimizing a 'surrogate' objective function using stochastic gradient ascent. Sampling happens by collecting a batch of samples from consecutive agent-environment interactions, called trajectories. Then the sampled trajectories will undergo an optimization process in order to have the best gradient updates. After a fixed amount of optimization iterations, the policy gets updated, and a new batch from the environment is taken for its next update. These optimization methods allow for a more coherent improvement of the policy, thus reducing variance. The use of trajectory samples can be seen as a way of using short-term memory that depends on the set horizon or batch. When setting a long horizon, the policy will improve fewer times than setting a short horizon. In general, a long horizon will result in better reward maximization, thus a better policy and a short horizon will result in higher sample efficiency, resulting in a faster learning agent. In this section, two policy optimization methods are presented, the first being the Trust Region Policy Optimization (TRPO) which is an algorithm that is based on a theoretical method called Trust Region Optimization in section 4.3.1. The second method is the Proximal Policy Optimization (PPO) in section 4.3.2, which is a method that approximates the idea and performance of TRPO.

4.3.1. Trust Region Policy Optimization

Conventional stochastic policy gradient methods and Approximate Dynamic Programming are known to be sample inefficient and unreliable when applied to high-dimensional control problems, as these methods perform one gradient update per data sample. A method that tackles these two main problems is the Trust Region Policy Optimization (TRPO) [48]. TRPO uses the principles of a second-order iterative numerical optimization method called Trust Region. The Trust Region method involves solving hard constraints that demarcate the optimization process. As such, a trust region is a local area in parameter space where the next optimization point (i.e., the current policy) is not far away from a specified point (i.e., old policy) in parameter space. The distance between these points in parameter space can be quantified using any distance measure. For TRPO, the Kullback–Leibler (KL) divergence is used as a distance measure that allows for robust and large updates. KL divergence is also known as relative entropy and is a metric of how a probability distribution is different from a reference probability distribution. TRPO used a shortcut to make the Trust Region Optimization more practical and applicable for on-policy reinforcement learning by taking the average of all the hard constraints. As a consequence of the practical implementation of the Trust Region method as an optimization constraint, the optimization process is stable and robust during the on-policy learning process. The sampling of trajectories reduces the variance of conventional stochastic policy gradient methods. The Trust Region optimization over the sampled trajectories allows for a stable and reliable policy gradient optimization method for stochastic policy gradient methods. TRPO can scale these advantages to high-dimensional control problems when a base stochastic policy gradient method is using deep neural networks as nonlinear function approximator.

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] \tag{4.1}$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \right] \right] \leq \delta \tag{4.2}$$

The algorithm first collects a sample trajectory, then averages over the samples in the trajectory and constructs the estimated objective in eq. (4.1) that needs to be maximized under the constraint in eq. (4.2) that constrains the size of the policy update, then as the last step, is to approximately solve the constrained optimization problem to update the policy's parameter vector θ. The objective function in the optimization problem can be seen as a 'surrogate' objective function, as this should be included in the cost-to-go function of a reinforcement learning method.

4.3.2. Proximal Policy Optimization

TRPO is a relatively complicated method which makes implementation less practical, and being a second-order iterative optimization method, it requires more computing time per sample. In addition, it does not allow neural network architectures with parameter sharing between value and policy function estimation, thus opting out for additional gains in sample efficiency. However, the sample efficiency and guarantee of monotonic improvement of TRPO are of great benefit to many continuous control problems. The Proximal Policy Optimization (PPO) algorithms [50] solve the problems identified with TRPO by emulating its ideas with a first-order iterative optimization method.

$$J^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right] \tag{4.3}$$

$$J^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right) \hat{A}_t \right) \right] \tag{4.4}$$

PPO emulates the idea of Trust Region by setting out a proximal region around an estimated policy probability ratio as in indicated eq. (4.3), where it uses the concept of importance sampling (also seen in DPG [56] in section 4.1.1) to obtain the expectation of samples gathered from an old policy under the current policy to refine its current policy. Equation (4.3) is the performance function of a Conservative Policy Iteration (CPI) method. This proximal region is set by the clipping ratio ϵ, as indicated in eq. (4.4), where the performance function of the CPI method can become the lower bound when the old policy has a higher advantage than the current policy. The stability of policy updates is ensured by not allowing for too large updates to be applied to the 'to be updated' policy. This is done by clipping the probability ratio $r_t(\theta)$ to the proximal region (1 - ϵ, 1+ ϵ). When taking the expectation over a sampled trajectory, then the upper clipping bound makes PPO not too greedy when having actions that have a large and positive advantage, whereas the lower clipping bound makes it not too conservative by avoiding actions with a large and negative advantage. An image to illustrate these clipping bounds is provided in fig. 4.1.

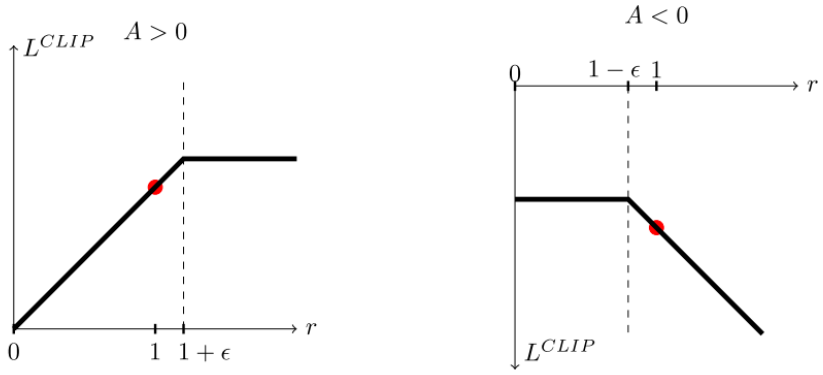

Figure 4.1: "Plots showing one term (i.e., a single timestep) of the surrogate function J^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that J^{CLIP} sums many of these terms" [50]. In the plot J^{CLIP} is indicated as L^{CLIP}.

Here one could argue that PPO is an off-policy method as it samples from the environment using a trajectory created by an old policy. However, policies learned with PPO, update their policy within one policy

iteration, so it is still an on-policy method as PPO learns its policy during interaction with an old policy that is not that different from the current or the to be updated' policy. This differs from the current policy and a target policy used in off-policy methods. So it could be said that PPO is a hybrid as it is in between an on-policy and off-policy actor method.

The PPO method distinct itself through its simplistic implementation as a first-order iterative optimization method that optimizes a surrogate objective with clipping probability ratios. The clipping of the probability ratios allows for stabilized and sample efficient learning, and exploration can be stimulated by augmenting the performance function with an entropy bonus. In addition, PPO allows for shared representations of the networks of the value and policy function this means that less learning is required as the knowledge is shared, So this allows for speed up in learning and results in a higher sample efficiency. A way to augment both value function network parameter sharing and an entropy bonus in a performance function is shown in eq. (4.5). Where VF stands for value function, S is the entropy bonus, c_1 and c_2 are coefficients that can be set based on their relative importance [1].

$$J_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[J_t^{CLIP}(\theta) - c_1 J_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right](s_t) \right] \tag{4.5}$$

On a final note, the author [50] implemented PPO with a Generalized Advantage Estimator (GAE) [49] that was inspired by the use of an advantage function in A2C as proposed in [37] and presented in section 4.1.2. The PPO with GAE was then tested on OpenAI environments that used the MuJoCo physics engine, where it showed good results for double inverted pendulum (having non-minimum phase behavior) as it showed much better sample efficiency than for TRPO with and without A2C. For the more complex and high-dimensional tasks such as manipulation and locomotion tasks, PPO also outperformed A2C, TRPO with and without A2C. This shows that the method is scalable to complex and continuous domains.

4.4. Hierarchical Design in (Deep) Reinforcement Learning

Hierarchical Design in Reinforcement Learning or often referred to as Hierarchical Reinforcement Learning (HRL) and refers to the field that tackles the curse of dimensionality by structuring and/or decomposing the state and/or action space of the agent's environment as such that the agent is able to learn with fewer samples than an agent without Hierarchical Design. Hierarchical Design can be expressed in a variety of ways by structuring and decomposing the underlying MDP. The decomposition of the underlying dynamics of an RL problem is done by decomposing it into multiple 'activities' followed by a structure that links the multiple activities in a coherent Hierarchical Design.

A few examples of these 'activities' from literature and their Hierarchical Design can be: the result of sub-policies defined in a sub-state space, this can be done with the Options framework [59] which allows the programmer to define sub-policies called Options and let them be structured under a policy over options. Or an additive combination of value functions with subgoals set by the designer, this can be done with the Hierarchical Abstract Machines (HAM) [16] framework that allows the programmer to construct multiple machines that manage action generation. Or by decomposing the value function and bounding the action space, this can be done with the MAXQ [43] framework that allows the programmer link the sub-tasks in a tree structure. Or by decomposing the problem with respect to the state space with each having their own value function and producing an explicit goal for the bottom-level policies this can be done with Feudal Reinforcement Learning [13] which allows the programmer to define multiple state-space where each level has a manager is operating on a more granular part of the state space, thus creating a funnel with increasing level of detail of the state space.

In the field of Hierarchical Reinforcement Learning, three highly regarded methods were identified [5] that were common in their reliance on the theory of Semi-MDP [59]. These methods are the Options [59], MAXQ [16], and Hierarchies of Abstract Machines (HAM) [43] framework. A fourth framework that was not in this review of Barto [5] – as it does not rely on the Semi-MDP but on the MDP as it is not a temporal abstraction but a state abstraction – is Feudal Reinforcement Learning [13]. These HRL methods are frameworks that require the programmer to decompose the problem into 'activities' this requires the programmer to have the domain knowledge to identify structures present in an environment. In fault-tolerant flight control, this structure is not always known. In addition, it is preferred for the general applicability of the control design to have an HRL method that can learn to decompose the environment's hierarchical structure.

In this review, the focus is on end-to-end learning of Hierarchical Design through policy gradient reinforcement learning. The HAM and MAXQ frameworks have successful implementations with Q-Learning [43] [16], though they do not have any well-cited implementations with policy gradients for continuous con-

trol problems. In addition, these methods require prior knowledge of the environment and its hierarchical structure. The authors of these methods identified the difficulty that comes with establishing relationships between the defined 'activities', and the hierarchical structure of an environment [43] [17]. In this , the HAM and MAXQ frameworks will not be elaborated as these do not have a proven learning method for their policy gradient application and – at the time of writing – no end-to-end learning variant of these methods were found in the literature. For more information about the HAM and MAXQ methods, the reader is advised to read [43] and [16], respectively.

The focus of this section will be on mainly hierarchical policy gradient reinforcement learning techniques, as policy gradient techniques is found to be the most suitable technique for continuous and complex real-world problems, though to render these techniques to be applicable for flight control it needs to be more sample efficient while remaining adaptive. In increasing the sample efficiency of policy gradient methods, a look towards Hierarchical Design for policy gradient techniques might provide a solution for sample efficient and adaptive control for large continuous state spaces [20]. A Deep Reinforcement Learning algorithm with Hierarchical Design, Hierarchical Deep Deterministic Policy Gradient (h-DDPG) will be presented in section section 4.4.1 and a variation called Hierarchical Intermittent Deep Deterministic Policy Gradient (HI-DDPG) in section 4.4.2. In section 4.4.3 a policy gradient implementation of the Options framework, the Option-Critic Architecture, will be presented.

4.4.1. Hierarchical Deep Deterministic Policy Gradient

A policy gradient method that combines Actor-Critic Design with Hierarchical Design is the novel Hierarchical Deep Deterministic Policy Gradient (h-DDPG) introduced by [76] that is designed to reuse learned basic skills. The Hierarchical Design in this method decomposes the problem into a variety of basic skills that can be learned by Actors of the agent in an end-to-end fashion through a multi-DDPG architecture introduced in [75]. The multi-DDPG is a single Critic and multi-Actor method that allows for each Actor network to learn independently of each other. As a result, basic skills can be learned with different requirements and movements. A basic skill is thus an Actor with its own subgoal when it fulfills this subgoal then the Actor is able to execute a basic task. The multi-Actor architecture in its turn is structured under a dual-Critic structure, making the Hierarchical Design complete. The duel-Critic enables the compounding of basic skills to solve for compounded tasks. The duel-Critic has a basic Critic and a meta Critic. The basic Critic is the Critic that is situated in the multi-DDPG and is tasked with solving basic tasks that result from the mechanics behind the reward vector that is set by the programmer. At the hierarchical level at which the basic Critic is situated, the basic skills are able to solve for basic tasks that are not goal-oriented such as rotating a wheel or bending a joint in a specific direction. The meta Critic focuses on learning compound skills by compounding the basic skill as a result, the agent is able to solve for compound tasks that are goal-oriented. A compound task would be to reach a point (x, y, z) in three-dimensional space by using basic skills. An overview with all the discussed components is provided in fig. 4.2.

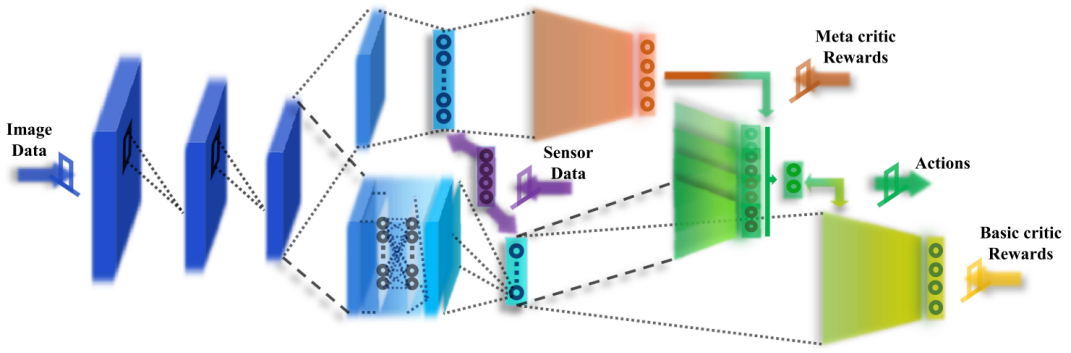

Figure 4.2: A Hierarchical Design with Deep Deterministic Policy Gradient, adapted from [76]. The basic Critic layer, indicated in yellow, needs a reward function – provided by the programmer – that sets the subgoals. The meta Critic layer also needs a reward function that sets the goal of the agent. Note that the meta Critic receives a more abstract set of states than the basic Critic. So two levels of image feature abstraction were established to be consistent with the whole Hierarchical Design. Also, note that two kinds of data are used as state information.

The advantage of this Hierarchical Design is that the skills can be reused for complex tasks and reduces

learning effort for these complex tasks. Another advantage is that the multiple basic skills can be learned at the same time and that the learning process for both basic skills and compounded skills are learned in one process and time scale. At the same time, this is a drawback of this method, as both hierarchies are operating on the same time scale this does not allow for temporal abstraction and decomposition of the environment.

The h-DDPG method acquires high-level skills – compound skills – from a finite set of basic skills. The method adopted a discrete action space on the higher level for combining these basic skills into compound skills. As a result of the discrete domain on the high-level, low-resolution compound skills are created that might not offer much flexibility for continuous motion tasks. In [51] it was found that h-DDPG was not able to discover new skills during the joint learning of the hierarchical modules. In addition, the h-DDPG method can have frequent switching of basic skills that can reduce the consistency, smoothness, and might harm the performance.

The concept of the Meta-Critic in Hierarchical Reinforcement Learning presented here; is taken a step further by [74] to end-to-end knowledge transfer, where the proposed Hierarchical Meta-Critic Networks (HMCN) is able to utilize the basic Critic and Meta-Critic networks with a task specified network to distill meta-knowledge from distribution of related tasks. The meta-knowledge can then be utilized to supervise a sect of actors that can solve any set of specified tasks.

4.4.2. Hierarchical Intermittent motor control with Deep Deterministic Policy Gradient

A method that is related to the h-DDPG (see section 4.4.1), in regard to the components and motivation, is the Hierarchical Intermittent motor control with Deep Deterministic Policy Gradient (HI-DDPG) [51] method. HI-DDPG aims to solve identified challenges in [76] and in the field of reusing learned sub-policies. Where the compounding of basic skills could result in powerful compound skills but was not able to let the meta critic set subgoals for the multiple actors that were learning basic skills. As a result, the programmer was tasked to set these subgoals. The HI-DDPG alleviates the programmer from the task of setting these subgoals by automatically generating the subgoals by employing two DDPG methods stacked on top of each other. Where the top-level DDPG sets a sequence of goals for the DDPG method on the lower level, thus these goals then become the subgoals in regard to the whole architecture. This architecture allows HI-DDPG to modulate the basic skills using a top-down approach, whereas the h-DDPG had a bottom-up approach as the compound skills were defined by the basic skills. The similarities can be drawn between Feudal Reinforcement Learning [13] where a high-level manager sets subgoals for the mid-level manager below him. Though HI-DDPG achieves temporal abstraction as the state space remains equal for each level, whereas Feudal Reinforcement Learning achieves state-space abstraction as each level of the manager decreases it operates in a more granular state-space.

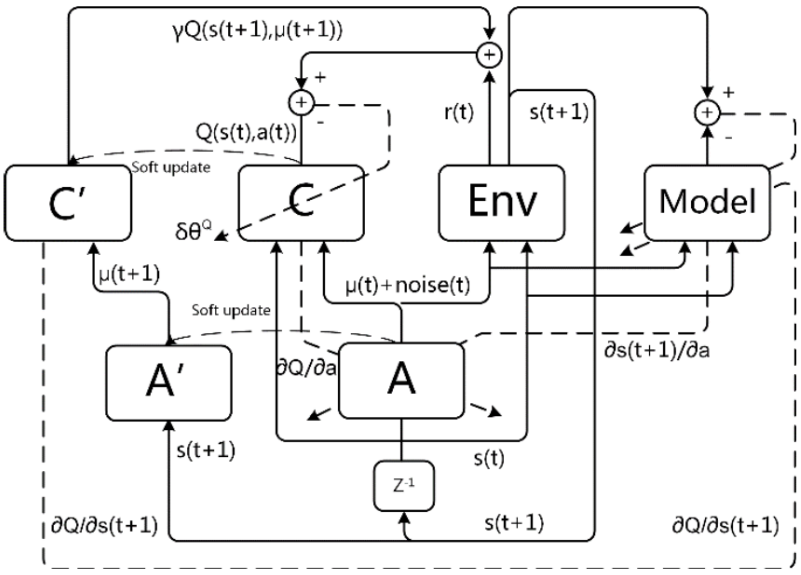

Figure 4.3: A Hierarchical Design with Deep Deterministic Policy Gradient and Intermittent control with a model-based gradient as an extension, adapted from [51]. Note that there is only one low-level Actor, whereas h-DDPG [76] had multiple low-level Actors.

The temporal abstraction capability of HI-DDPG is enabled through its low-level and high-level Actor that is operating at different time scales. In addition, the high-level Actor operates in the continuous domain (as opposed to h-DDPG), where it provides continuous-valued goals and duration. These two outputs are then provided to the low-level DDPG. The low-level Actor then adjusts its basic skills according to the provided goals within the time horizon. As a result, the HI-DDPG is able to jointly learn control policies on two levels that are capable of completing composite tasks. An overview of the architecture is provided in fig. 4.3.

Experiments performed on the Puckworld environment showed better exploration behavior of HI-DDPG than for h-DDPG and its non-hierarchical counterpart. The exploration behavior of HI-DDPG suggests that it has the capability to find a global optimum [51] and thus was able to achieve the highest reward in this experiment. In addition, HI-DDPG was found to be computationally efficient, though the agent needed over 2500 episodes to perform better than the flat DDPG and needed 5000 episodes to perform better than h-DDPG. The HI-DDPG without model-based gradient performed even worse, though this indicates that their model-based gradient is able to accelerate convergence and learning of the HI-DDPG method.

4.4.3. Option-Critic Architecture

The Option-Critic architecture [4] automatically decomposes its problem domain into a set of temporally extended actions, which are called Options [59]. A way of viewing these Options is to see them as a set of sub-policies with each having its own variable time scale in which the Option operates depending on its initiation state and termination condition. For the Options in [59] the initiation state and termination condition needs to be set by the programmer, whereas with the Option-Critic, the initiation set is omitted, and the termination condition is learned. The aforementioned sub-policies are also known as intra-option policies π_ω and the variable time scales are now set by their respective termination function β_ω. The Options are selected and structured by the main policy called the policy over options π_Ω. The Option-Critic architecture learns these Options in an end-to-end fashion by applying the policy gradient theorem to the intra-option policies, termination functions, and policy over options, in order to optimize a task that is encoded in a reward function. The name Option-Critic stems from its similarities with the Actor-Critic Design, where the intra-option policies with its termination function act as multiple Actor networks that gets selected by a higher level Actor network (policy over options). The intra-option policy, termination function, and policy over options gets feedback and update information from their Critic networks that estimate the state-action value function Q_U of the options and the advantage function of the policy over options A_Ω, as depicted in section 4.4.3.

The advantage of the Option-Critic architecture, according to the experiments performed by [4] with the four-room environment, is the ability to learn Options from scratch without slowdown of learning when compared to non-hierarchical methods. The method is versatile, as it allows to be combined with other non-hierarchical reinforcement learning methods and results in comparable to improved learning speed while learning multiple sub-policies. The sub-policies and termination functions that are part of the learned Options, when visualized, allow for better interpretability of learned control law. In addition, the Option-Critic can act as a baseline for transfer learning as these learned options can be reused for other similar domains. Behavior that shows that it can act as a baseline for transfer learning is the proven ability to recover from sudden changes made in the environment. Also, the Option-Critic can fit in learning with different base deep reinforcement learning algorithms. This shows that Option-Critic is a general framework and also is able to generalize for changes made to the environment.

As opposed to h-DDPG [76] and other HRL methods, the Option-Critic only requires the programmer to specify the number of desired options. The Option-Critic framework automatically generates subgoals for the low-level policies and allows for more control of specialization of low-level policies by setting pseudo-reward functions. In addition to action space decomposition, the state space can also be decomposed by setting initiation states for each low-level policy. Next to the fact that Option-Critic does not require subgoals. It also does not require additional rewards, demonstrations, multiple problems, or any other special accommodations.

Though the Option-Critic still has open-ended problems, one of them being that all Options are converging to the same intra-option policy behavior if learning is set to a continual learning setting. This problem can be derived from the goal of the Option-Critic method, which is to learn Options that maximize the expected return in the current task, but as a consequence of optimizing for the return, the termination gradient tends to shrink Options over time. This is expected since in theory, primitive actions should be sufficient for solving any MDP. This problem seems to be common in the field of HRL, as it was also identified by [19] with their Stochastic Neural Networks for Hierarchical Reinforcement Learning (SNN-HRL). They aimed to learn useful skills during the pre-training environment and then leverage the acquired skills for faster learn-

ing during operation and interpretability. A similar idea is behind the Option-Critic framework, and just like the Option-Critic framework, the SNN-HRL also encountered the same problem as the acquired skills could eventually converge to the same policy behavior the authors of [19] aimed to solve this phenomenon with an information-theoretic regularizer, specifically they used a mutual information (MI) bonus, using an entropy function, that was added to the reward function. The use of the MI bonus proved successful for diverse skill learning, but does not always boost performance. So most likely, a tradeoff should be made here.

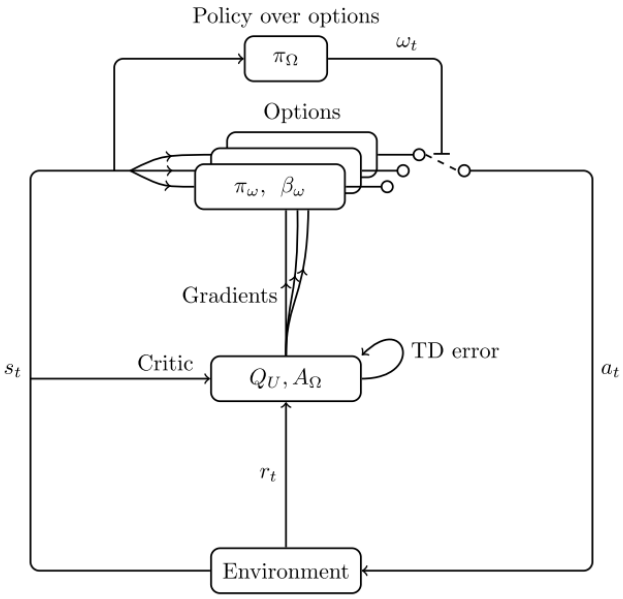

Figure 4.4: An overview of the Option-Critic architecture, image was adapted from [4]

adInfoHRL [42] also aims at retaining Options for an extended period and by keeping the specialization through mutual information (MI) maximization, where this method adds an extra component called the advantage-weighted importance. The advantage-weighted importance makes use of the multiple modes present in the advantage functions, which allows adInfoHRL to divide the state-action space instead of only the state space as for [32]. Though its proven success is built on the dependence of advantage functions with multiple modes and a deterministic policy gradient method to learn the Options. The adInfoHRL should be applicable to stochastic gradient policies, but the author admits that this still requires further investigation [42].

Another way of solving the problem of intra-option policy behavior convergence, is by fixing the intra-option policy and as an added benefit have a much higher sample efficiency. A method to do this is a hybrid form of end-to-end learned Options called the Option-Interruption architecture [32], where the Interruption mechanism is mainly based on learning a termination function as is the case for the Option-Critic architecture. The large difference being that the Option-Interruption method allows for hand-crafted intra-option policies to be included, enabling established control laws to be incorporated. As a consequence, the method's search space is considerably reduced and allows for a higher sample efficiency when learning these intra-option policies. Though in some circumstances, such as in flight recovery, incorporating human knowledge is complicated and might not be sufficient, thus the author [32] suggests combining learned Options with human-suggested Options.

The results presented in [4] were from environments that did not have a continuous state space but a discrete action space. Research in the strength of the Option-Critic for continuous state and action space was conducted [29] using the PPO method with a deliberation cost [22] that should prevent Options from switching too much. The results indicate that adding deliberation cost results in faster learning and that the Option-Critic with PPO has proven successful for continuous action space, though more research is needed for setting a good deliberation cost. This implementation of the Option-Critic showed significantly better performance for environments with clear compositionality. In flight control, this compositionality might be apparent in the distinction between faster and slower dynamics present in aircraft.

The Option-Critic is used in an active research field and has many directions such as end-to-end Option

learning, transfer learning, meta-learning, and multi-task learning. The Option-Critic can be extended to multi-level hierarchies [47], or stacked on top of each other to create a Double Actor-Critic architecture [77] to learn options that also forms a general framework for policy optimization algorithms such as PPO.

4.5. Conclusion

The goal of this chapter is to provide an answer to the second research question: **What are the current state-of-the-art reinforcement learning methods that are suitable for flight control?** This question is answered by answering its sub-questions that are restated here:

a) ***What is the most suitable state-of-the-art policy gradient reinforcement learning technique?***

A distinction of the reviewed policy gradient reinforcement learning (PGRL) methods are made on the basis of their focus of solving stable learning for high-dimensional state spaces. Deep Reinforcement Learning in section 4.1 focusses on computational efficiency, sample efficiency, and high-dimensional continuous problems, whereas in Approximate Dynamic Programming in section 4.2 focusses on designing online learning methods for uncertain dynamic systems. Both fields face challenges with good exploration and at the same time having fast convergence through high sample efficiency. The policy optimization methods in section 4.3 intently focus only on the challenge of stable learning for large continuous domains while having good exploration and high sample efficiency. From all non-hierarchical PGRL frameworks, the Policy Proximal Policy (PPO) method implemented with an Actor-Critic design and Generalized Advantage Estimation (GAE) function is found to be the most suitable to achieve stable and sample efficient learning for control in a high-dimensional, continuous and stochastic domain. The aforementioned has been proven for MuJoCo physics environments but has not been investigated for flight control of passenger carrying aircraft. Thus an initial investigation in the properties of the PPO method with GAE and Actor-Critic Design is conducted in this research.

The PPO method allows for stochastic policies to be sample efficient and retain their on-policy learning capability, whereas Deep Deterministic Policy Gradient (DDPG) exchanges the stochastic policy for more sample efficiency. The drawback is that the DDPG method is an off-policy method. As a consequence, the method is not adaptable in an online setting. The lessons learned from the DHP and IDHP method can be applied to the final control design. One of the learned lessons is that both methods estimate the gradient of the value function directly. Additionally, IDHP utilizes an online incremental model estimator to enable online-only learning. Though the DHP and IDHP methods are learning a deterministic policy and no existing implementation and analysis were performed on PPO combined with IDHP methods. In addition, the mathematical compatibility of these methods is not yet proven. As a consequence, it is preferred to keep the gained insights of the DHP and IDHP as a possible add-on for final control design.

A benefit of IADP methods is that they can be applied as online-only training methods that provide a proven solution for flight control. The online-only training capability in IDHP is proven to work for small state spaces where it can provide for control of states with a short time separation. As a result, IDHP was able to perform full-body rate control by using two IDHP agents. One agent one for longitudinal and the other for lateral control. Though, IDHP for full altitude control with an extended state space showed slower learning. In this regard, offline training methods used in Deep Reinforcement Learning such as Proximal Policy Optimization, are able to find more complex control behavior, and when combined with Hierarchical Design, can have faster learning. The aforementioned characteristics are mainly based on achieved successes in complex motion control such as locomotion and manipulation tasks. These demonstrations show more promising performance in terms of adaptivity, generalizability, and possibly sample efficiency than online-only training methods such as IDHP.

b) ***What is the most suitable state-of-the-art hierarchical policy gradient reinforcement learning technique?***

Hierarchical Design in Deep Reinforcement Learning in section 4.4 decomposes a complex problem into multiple activities connected by a hierarchical structure. The idea behind Hierarchical Design is to abstract a complex problem, and as a consequence, learn faster than methods without Hierarchical Design. The reviewed methods are selected on either of two aspects: Hierarchical Design frameworks that have proven

implementations with policy gradient reinforcement learning. Secondly, the possibility to have an end-to-end learning algorithm that requires minimal prior knowledge. The Option-Critic architecture is found to be the most suitable as the temporal abstraction capability provides an opportunity to link this to the time scale separation present in flight control. The method's ability to recover from sudden changes made in the environment is beneficial for stochastic phenomena such as turbulence during flight or sudden actuator failure. An added benefit is that this method fits with any base policy gradient reinforcement learning method i.e., Proximal Policy Optimization. The versatility of the Option-Critic allows for applying domain knowledge by setting pseudo-rewards or initiation states. Though, demonstrations have shown that this is not required to have good performance. In addition, Option-Critic architecture is scalable to large domains as it interfaces with existing deep reinforcement learning algorithms and has higher sample efficiency for environments with high compositionality that will be of great benefit for a flight controller during online operation.

The Hierarchical- and Intermittent Deep Deterministic Policy Gradient (h-DDP and HI-DDPG) have shown that hierarchical policy gradient reinforcement learning methods are capable of learning advanced complex control behavior by learning and reusing basic skills. The deterministic setting allowed for a replay buffer to be used, but at the same time, required these methods to learn in an off-policy setting. h-DDPG, HI-DDPG, and Option-Critic have the ability to generalize well to changes made in an environment, but h-DDPG and HI-DDPG are not general frameworks that can interface with both off-policy and on-policy methods. The Option-Critic architectecture is a general framework for Hierarchical Design and allows for straightforward application for flight control in an on-policy fashion.

In conclusion, the use of Option-Critic architecture in conjunction with the Proximal Policy Optimization algorithm should provide a state-of-the-art hierarchical policy gradient reinforcement learning technique that achieves sample efficient, adaptive, and continuous control for high-dimensionality and stochasticity present in aircraft dynamics.

5

Preliminary Analysis

From the literature study a most promising hierarchical policy gradient technique, the Option-Critic architecture, was selected that will be tested in this chapter for its beneficial properties which are adaptivity and sample efficiency for large state spaces. The testing will be done by comparing a hierarchical policy gradient reinforcement learning technique to its non-hierarchical variant in a set of experiments.

The aim of this preliminary analysis is to verify and possibly find additional benefits of hierarchical methods, to be precise this analysis should provide an answer to research question three, as formulated in section 2.2 and restated here:

3. **What are the benefits of the proposed hierarchical policy gradient technique over the proposed non-hierarchical policy gradient technique on a mass-spring-damper system when exposed to unexpected changes?**

   (a) *What is the performance regarding adaptivity?*

   (b) *What is the performance regarding sample efficiency?*

   (c) *What is the performance regarding reference tracking?*

In answering the research question, a more in-depth look will be taken into the algorithms of the proposed hierarchical and non-hierarchical agent in section 5.1. Secondly, the problem setup for the agents in section 5.2. Thirdly, the experiment setup of the environment of the agents in section 5.2. Fourthly, the results regarding adaptivity, sample efficiency, and reference tracking in section 5.3. Lastly, the conclusion in section 5.4.

5.1. Agents

In answering sub-research question three in chapter 6, it was found that the Proximal Policy Optimization method implemented with an Actor-Critic Design and GAE (see section 4.1.2 and section 4.3.2) is the most suitable non-hierarchical method. The resulting agent will be known as PPO agent and its algorithm is presented in algorithm 1. Consecutively, in answering sub-research question four in chapter 6, it was found that the Option-Critic architecture is the most suitable hierarchical policy gradient method and will be implemented on top of the PPO agent implementation. The resulting agent will be known as the Proximal Policy Option-Critic (PPOC) agent and its algorithm is shown in algorithm 2. Lastly, the hyperparameter optimization method used for both agents will be elaborated in section 5.1.1.

Algorithm 1 Actor-Critic with Proximal Policy Optimization

Require:
agent hyperparameters α_θ, α_ϑ, γ, λ, ϵ, T
differentiable stochastic policy parameterization $\hat{\pi}(a|s,\boldsymbol{\theta})$
differentiable state-value function parameterization $\hat{V}(s,\boldsymbol{\vartheta})$

$s \leftarrow s_0$
while $s' \neq$ *terminal* **do**
  $a \leftarrow \hat{\pi}(a \mid s,\boldsymbol{\theta})$
  for iteration = 1, 2, ... **do**
    procedure RUN POLICY $\hat{\pi}_{\theta_{old}}$ IN ENVIRONMENT FOR T TIME STEPS
      take action a while in s and observe s', r
    end procedure
    procedure COMPUTE ADVANTAGE ESTIMATES $\hat{A}_1, \ldots, \hat{A}_T$
      $\delta_t \leftarrow R_{t+1} + \gamma \hat{V}(S_{t+1},\boldsymbol{\vartheta}) - \hat{V}(S_t,\boldsymbol{\vartheta})$
      $\hat{A} \leftarrow \delta_t + \gamma\lambda\hat{A}$
    end procedure
  end for
  procedure OPTIMIZE SURROGATE PERFORMANCE FUNCTION W.R.T. θ
    for K optimizer epochs with minibatches M **do**
      $\theta_\pi \leftarrow \theta_\pi + \alpha_{\theta_\pi} \frac{\partial J(\theta)^{\text{PPO}}}{\partial \theta_\pi}$
      $\vartheta_v \leftarrow \vartheta_v - \alpha_{\vartheta_v} \frac{\partial \delta_t}{\partial \vartheta_v}$
    end for
  end procedure
  $s \leftarrow s'$
end while

5.1.1. Hyperparameter Optimization

In general, reinforcement learning agents are sensitive to hyperparameters. The PPO and PPOC agents have the same hyperparameters, except that PPOC has an additional hyperparameter which is the number of options n. In order to get the best performance and to make the agents comparable, the same hyperparameter optimization (HPO) method is used for all agents. In order to study the behavior the Option-Critic, the number of options is not treated as a hyperparameter. Instead three settings are chosen for n, namely n=2, n=4, n=8. As a result of the settings of n, three different Option-Critic agents are created: PPOC-2, PPOC-4 and PPOC-8. The set of hyperparameters that need to be tuned are in table 5.1. So in total there are four agents with the same hyperparameters and hyperparameter optimization method. As a result this makes the tuning as fair and comparable as possible for the scope of this research.

Table 5.1: Hyperparameter value range for hyperparameter optimization using the TPE sampler. The values indicated without brackets are categorical values for a discrete distribution to select one of the values that is provided. The values with a bracket indicate the lower and upper value, in this range a uniform logarithmic and continuous distribution is employed to select a value. The hyperparameter values are taken from [23], [50], [9] and [26].

Hyperparameter	Value
Actor batch T	128, 256, 512, 1024, 2048, 4096
Clipping ratio ϵ	0.1, 0.2, 0.3, 0.4
Optimization epochs K	1, 5, 10, 20
Optimization batch size M	16, 32, 64, 128, 256
Learning schedule lr	linear, constant
Optimization step size α_θ	[1e-6, 5e-3]
Entropy coefficient S	[1e-8, 1e-4]

Optuna [2] is used as HPO method, because for its ease of use and optimization performance. Optuna is an API that uses the define-by-run principle, which allows the user to dynamically construct its hyperparameter space. After construction of its hyperparameter space, Optuna formulates the hyperparameter optimization by minimizing an objective function that takes a set of hyperparameters – from its hyperparameter space – as an input and returns its evaluation score. The objective function gets gradually built through

Algorithm 2 PPOC: Proximal Policy optimization with Option-Critic

Require:
agent hyperparameters α_{θ_π}, α_{θ_Ω}, α_{ϑ_β}, $\alpha_{\vartheta_\vartheta}$, γ, λ, ϵ, T, n
differentiable stochastic intra-option policy parameterization $\pi_\omega(a|s,\boldsymbol{\theta_\pi})$
differentiable termination function parameterization $\beta_\omega(s,\boldsymbol{\vartheta_\beta})$
differentiable stochastic policy over options parameterization $\pi_\Omega(\omega|s,\boldsymbol{\theta_\Omega})$
differentiable option-value function parameterization $V_\Omega(s,\omega,\boldsymbol{\vartheta_\vartheta})$

$s \leftarrow s_0$
$\omega \leftarrow \pi_\Omega(\omega \mid s,\boldsymbol{\vartheta_\vartheta})$ with softmax policy
while *episode* $\neq$ *terminal* **do**
  for iteration = $1,2,\ldots$ **do**
    procedure RUN POLICY π_ω^{old} IN ENVIRONMENT FOR T TIME STEPS
      $a_t \leftarrow \pi_\omega(a_t \mid s_t,\boldsymbol{\theta_\pi})$
      take action a_t while in s_t and observe s_{t+1}, r_t
      if β_ω terminates in s_{t+1} **then**
        choose new ω_{t+1} according to softmax $\pi_\Omega(\omega_{t+1} \mid s_{t+1},\boldsymbol{\vartheta_\vartheta})$
      end if
    end procedure
    procedure OPTIONS EVALUATION: COMPUTE ADVANTAGE ESTIMATES FOR T TIME STEPS
      $\delta_t \leftarrow r_{t+1} + \gamma V_\Omega(s_{t+1},\omega_t,\boldsymbol{\vartheta_\vartheta}) - V_\Omega(s_t,\omega_t,\boldsymbol{\vartheta_\vartheta})$
      $\hat{A}_\Omega^{GAE}(s_t,\omega_t) \leftarrow \delta_t + \gamma\lambda\hat{A}_\Omega^{GAE}(s_t,\omega_t)$
      $\hat{A}_\Omega^{\beta}(s_t,\omega_t) \leftarrow \sum_{\omega=0}^{k}\pi_\Omega(\omega_t \mid s_t,\boldsymbol{\vartheta_\vartheta})V_\Omega(s_t,\omega_t,\boldsymbol{\vartheta_\vartheta}) - V_\omega(s_t,\boldsymbol{\vartheta_\vartheta})$
    end procedure
    procedure OPTIONS IMPROVEMENT
      for $\omega = \omega^0,\omega^1,\ldots,\omega^k$ **do**
        $\theta_\pi^{old} \leftarrow \theta_\pi$
        for K optimizer epochs with minibatches M **do**
          $\theta_\pi \leftarrow \theta_\pi + \alpha_{\theta_\pi}\frac{\partial\hat{J}^{\text{PPO}}(\theta_\pi)}{\partial\theta_\pi}$
          $\vartheta_\beta \leftarrow \vartheta_\beta - \alpha_{\vartheta_\beta}\frac{\partial\beta_\omega(s_{t+1},\boldsymbol{\vartheta_\beta})}{\partial\vartheta_\beta}\hat{A}_\Omega^{\beta}(s_t,\omega_t)$
          $\theta_\Omega \leftarrow \theta_\Omega + \alpha_{\theta_\Omega}\frac{\partial\log\pi_\Omega(\omega_t|s_t,\boldsymbol{\vartheta_\vartheta})}{\partial\theta_\Omega}\hat{A}_\Omega^{GAE}(s_t,\omega_t)$
          $\vartheta_\vartheta \leftarrow \vartheta_\vartheta - \alpha_{\vartheta_\vartheta}\frac{\partial(G-V_\Omega(s_t,\omega_t,\boldsymbol{\vartheta_\vartheta}))^2}{\partial\vartheta_\vartheta}$
        end for
      end for
    end procedure
  end for
end while

its interactions with the agent and its hyperparameter space. A set of hyperparameters is statistically sampled based on the history of previously evaluated hyperparameters.

This sampling process in Optuna is done by a sampler. Optuna allows for a variety of samplers and pruners to be employed for its optimization process. The employed environment model is not compatible with the pruners that are provided by Optuna, thus the pruner capabilities are not used. The Tree-Structured Parzen Estimator (TPE) sampler method is employed for the hyperparameter optimization. TPE is a sequential model-based optimization (SMBO) method. These methods approximate the performance of hyperparameter by sequentially constructing models based on historical measurements [6]. From these models, it subsequently choose new hyperparameters to test. TPE distinct itself by its ability to sample for categorical variables and its runtime of each iteration during the optimization process. TPE's iteration runtime scales linearly in historical measurements and linearly in the number of hyperparameters that are part of the optimization. As opposed to using a Gaussian Process Approach which scales cubically in the number of hyperparameters. Thus TPE was chosen as a suitable method for a large hyperparameter space and for scarce computational resources [7] as is the case for this research.

5.2. Experiment setup

The experiments are conducted on a mass-spring-damper system as elaborated in section 5.2.1. The reward function that is used to learn the agent the desired behavior is found in section 5.2.2. The training setup can be found in section 5.2.3. The setup for testing adaptivity can be found in section 5.2.4. The evaluation of sample efficiency can be found in section 5.2.5.

5.2.1. Environment setup

In order to find differences between these agents, different control tasks in different environments were setup using an extendable mass-spring-damper-N system (MSD-N). The letter N is replaced by an integer defining the number of serially coupled MSD systems, fig. 5.1 provides an example of a MSD-3 system.

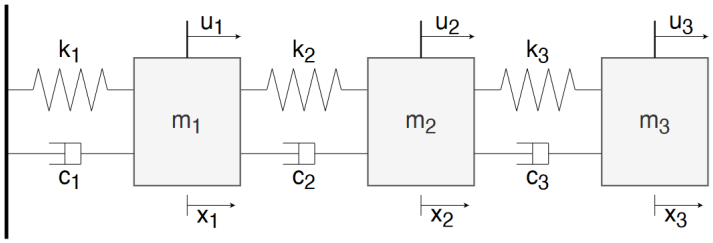

Figure 5.1: MSD-3, a mass-spring-damper system with three masses. The image is from [10].

The need for testing on multiple environments stems from that algorithm performance can vary across environments and the best performing algorithm across all environments is not always clear. A way to do this is to have an environment that can be increased in complexity. So an extendable MSD system was used for the experiments of this research. In addition, the extendable MSD system has similar dynamics and provides for testing in different settings that can approximate similar control complexity that is found for flight control. Working directly with the aircraft model would have resulted in less insight into the agent. Since aircraft dynamics are more complex and less intuitive thus aircraft model complexity would have increased the complexity of analyzing the results that provide understanding of agent behavior.

The approach for testing agent behavior for simplified flight control conditions is given in table 5.2. The experiments are numbered and these numbers will be used throughout the results. Each experiment is defined by its environment, number of tasks, number of actions and equivalence to flight control. The environment varies with the number of masses in the mass-spring-damper system. The number of tasks refers to how many states are being tracked. The task that must be performed by the agent is a reference tracking task for a specific state. In the case that only one task is set for the experiment then this will be a reference tracking task of either the velocity or position state of the first mass m_1 as depicted by fig. 5.1. The reference signal is a sinus. If the experiment has two tasks, then an reference tracking task will be added to the system. This second task has a zero reference signal and it will either track a velocity or position state of mass two m_2. The number of actions refers to how many input variables are accessible for the agents. The equivalence refers to

how the results from the experiment might give insight to either longitudinal or lateral control. In total there were 18 experiment combinations, only four have been selected that were judged as the most relevant for the preliminary analysis.

Table 5.2: Overview of experiment setups that is used to compare the agents.

Experiment	Environment	Number of tasks	Number of actions	Equivalence
1	MSD-1	1	1	Longitudinal control
2	MSD-2	2	2	Lateral control
3	MSD-3	1	1	Longitudinal control
4	MSD-3	2	2	Lateral control

Experiment 1 and 2 emulate the minimum amount of state and action space for its equivalence. Experiment 3 and 4 emulate a higher dimensional state space for flight control. In general lateral control is a more complex form of control than longitudinal control. In the case of these MSD systems this has been emulated by providing lateral control to have two tracking tasks, making the control task more complex for the agent. The setup of experiment 1 to 4 should provide results that give insight into the benefits of hierarchical reinforcement learning.

When setting up the tasks, the $Q^{R^{\text{lon}}}$ and $Q^{R^{\text{lat}}}$ matrices in eq. (5.3) (to be presented in section 5.2.2) remain the same for all longitudinal and lateral tasks. An example of a P matrix for a MSD-3 system is provided in eq. (5.1) and eq. (5.2). Where the p_{ij} and v_{ij} values can be set to either 0 or 1. When choosing for a position tracking task then the p_{ij} have to be set to 1 and the v_{ij} set to 0, vice versa for velocity tracking.

$$\boldsymbol{P}^{\text{MSD-3}} = \begin{bmatrix} p_{11} & 0 & 0 & v_{11} & 0 & 0 \end{bmatrix} \quad Q^{R^{\text{lon}}} = \begin{bmatrix} 1 \end{bmatrix} \tag{5.1}$$

$$\boldsymbol{P}^{\text{MSD-3}} = \begin{bmatrix} p_{11} & 0 & 0 & v_{11} & 0 & 0 \\ 0 & p_{22} & 0 & 0 & v_{22} & 0 \end{bmatrix} \quad Q^{R^{\text{lat}}} = \begin{bmatrix} 1 & 0 \\ 0 & 100 \end{bmatrix} \tag{5.2}$$

The value in eq. (5.1) was found by an trial-and-error process and proved to provide the best tracking performance and learning for the given experiment setups. The found value was re-used for eq. (5.2) and then accordingly tuned for the second tracking task which is 100 times larger than the value for the first tracking task.

5.2.2. Reward function

For continuous optimal tracking control a quadratic cost-to-go function implementation is taken from [25] and adapted for extra control on the magnitude of actions as can be seen in eq. (5.3). [25] based the implementation on the implementation of [80], where it has been adapted to a incremental variant of a Approximate Dynamic Programming (iADP) method where no time-scale separation is assumed and defines an one-step cost function, which is also expressed in eq. (5.3). This reward function will enable H_∞ control which is required as this flight controller is designed to be adaptive throughout its complete lifetime. In addition this reward function provides the ability to track continuous MIMO systems.

$$r_{t+1} = r\left(s_t^R, s_{t+1}\right) = -\left[\boldsymbol{P}s_{t+1} - s_t^R\right]^T Q\left[\boldsymbol{P}s_{t+1} - s_t^R\right] \tag{5.3}$$

Where P is the state selector which defines which states are selected for tracking, s_t being the state vector at time t, s_t^R being the reference signals, Q being the weighting matrix where the magnitude of each entry determines the priority of tracking with respect to each entry.

$$s^{\text{MSD-2}} = \begin{bmatrix} x_1 & x_2 & \dot{x}_1 & \dot{x}_2 \end{bmatrix}^T \quad s^{R^{MSD-2}} = \begin{bmatrix} \dot{x}_1^R & \dot{x}_2^R \end{bmatrix}^T \quad a^{\text{MSD-2}} = \begin{bmatrix} F_1 & F_2 \end{bmatrix}^T \tag{5.4}$$

Where $s^{\text{MSD-2}}$ is an example of a state vector for a mass-spring-damper system with two masses, where x_1 and $\dot{x}_1$ are the position and velocity of mass 1, respectively. Reference state vector $s^{R^{MSD-2}}$ contains the reference signals for each velocity of a mass and $a^{\text{MSD-2}}$ contains the control input of each mass, in this case it is a force input.

$$\boldsymbol{P}^{\text{MSD-2}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Q^{R^{\text{MSD-2}}} = \begin{bmatrix} q_{11} & 0 \\ 0 & q_{22} \end{bmatrix} \tag{5.5}$$

$$\boldsymbol{P}^{\text{MSD-3}} = \begin{bmatrix} p_{11} & 0 & 0 & v_{11} & 0 & 0 \\ 0 & p_{22} & 0 & 0 & v_{22} & 0 \end{bmatrix} \quad Q^{R^{\text{MSD-3}}} = \begin{bmatrix} q_{11} & 0 \\ 0 & q_{22} \end{bmatrix} \tag{5.6}$$

Equation (5.5) provides the matrices where the settings of the reward function can be set. In this case the P matrix is set for velocity tracking control. During this research the values in Q^R will be tuned for tracking performance.

5.2.3. Reward shaping and termination of training

The shape of the reward function is set by the values in the Q matrix as given in eq. (5.5) for a MSD-2 environment. First the agents will perform a velocity reference tracking task followed by a position reference tracking task on the same environment. From the first task, lessons are learned on how to shape the reward for the position tracking task. If the velocity tracking task was performing well, then the Q matrix for the velocity tracking will be re-used for the position tracking task. This process will start with the simplest environment which is a mass-spring-damper system with one mass referred as MSD-1 in table 5.2. This process repeats itself for increasing mass element for the MSD system. In principle each environment and task requires its own reward settings. For the purpose of this preliminary research the reward settings are kept as simple as possible by re-using the previous setting, if possible. Since the focus of this research is to compare a non-hierarchical agent with an hierarchical agent as to prove the value of a hierarchical architecture over a non-hierarchical one.

5.2.4. Adaptivity to changing environment dynamics

Adaptivity is tested by changing the internal dynamics of a mass-spring-damper system. The change in dynamics is done by changing the sign of either the spring or damping constant this depends on the tracking task. The change of sign will be from positive to negative this should make the system unstable and provide insight into how well the agents can adapt. The constant that is expected to be the most critical constant for the reference tracking task are the constants for the first mass m_1. So the spring and damping constant of m_1 will be changed during the adaptivity test.

There are two kind of tests, one where the spring constant is changed from $k_1 = 4$ to $k_1' = -1$, this will be called the k-test. The second test where the damping constant is changed from $c_1 = 3$ to $c_1' = -1$, this will be called the c-test. To have an overview where these coefficient are in the complete system, see fig. 5.1. In the case of MSD-3, the second test did not result in an unstable system so a third test was designed for MSD-3, where $c_1 = 3$ was changed to $c_1' = -1.5$ this will be called the $c_{-1.5}$-test. An overview of the eigenvalues of MSD-1, MSD-2 and MSD-3 are provided in table 5.3, table 5.4 and table 5.5, respectively. In the tables it can be seen that the introduced change to the constants result in unstable systems, as there are positive eigenvalues present. In the case of MSD-3 the new eigenvalues introduce large oscillatory behavior.

Table 5.3: Overview of eigenvalues for MSD-1 with the eigenvalues during nominal condition and the eigenvalues after their respective adaptivity test. The tests don't accumulate on top of each other. The eigenvalues of the tests are the result of executing only one test.

phase	λ_1	λ_2
nominal	-1.734	-5.766
k-test	+0.320	-7.820
c-test	+1.250+2.905j	+1.250-2.905j

Table 5.4: Overview of eigenvalues for MSD-2 with the eigenvalues during nominal condition and the eigenvalues after their respective adaptivity test. The tests don't accumulate on top of each other. The eigenvalues of the tests are the result of executing only one test.

phase	λ_1	λ_2	λ_3	λ_4
nominal	-19.716	-0.962+1.134j	-0.962-1.134j	-0.860
k-test	-20.256	+0.298	-1.019	-1.523
c-test	-12.032	+0.212+1.857j	+0.212-1.857j	-0.892

5.2.5. Sample efficiency

Initial sample efficiency runs were performed for a few experiment setups. In these runs the adaptivity test was not activated, thus the agent was learning for 240k time steps. A problem with this learning setup for the

Table 5.5: Overview of eigenvalues for MSD-3 with the eigenvalues during nominal condition and the eigenvalues after their respective adaptivity test. The tests don't accumulate on top of each other. The eigenvalues of the tests are the result of executing only one test.

phase	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
nominal	-27.687	-7.878	-2.218	-0.399+1.174j	-0.399-1.174j	-0.168
k-test	-27.688	-9.571	-0.807+1.324j	-0.807-1.324j	+0.292	-0.168
c-test	-27.673	-0.416+3.979j	-0.416-3.979j	-0.038+1.295j	-0.038-1.295j	-0.168
$c_{-1.5}$-test	-27.672	+0.165+3.838j	+0.165-3.838j	+0.005+1.349j	+0.005-1.349j	-0.168

agent is that an agent might require more time steps to reach a higher reward or less time steps to converge to a reward level. Thus an agent might require having a varying amount of time steps to learn compared to other agents. A solution to provide the agent with varying time steps to learn is to provide it with an automatic termination rule. The purpose of this rule is to terminate the whole learning process when a specific reward level or error has been reached. An advantage of this automated method of terminating learning is that it makes the runs easy to compare to each other by just looking at how many time steps an agent needs to learn for a specific reward level. A few drawbacks of using this termination rule, a priori knowledge is required to set the reward error or level in the case of this experiment setup it might require too much tuning. Specifically it would require tuning for both velocity and position tracking for each experiment. Another drawback is that the results cannot be easily compiled into one graph using confidence intervals or interquartile range.

So the total time steps was affixed for the sake of simplicity when compiling numerous runs into one graph and sample efficiency is evaluated by using the learning curves coming from the adaptivity tests for each experiment see fig. 5.14, fig. 5.15, fig. 5.16 and fig. 5.17. Though fixing the total time steps when looking at sample efficiency of agents comes with their own disadvantages. One of them is that a limited view of the agent's performance is portrayed by the graphs, as the learning of the agent is arbitrarily cut off. As a possible result the agent will not have converged to its reward level.

Luckily this does not have to a problem in answering the sub-research question. The focus here will be on minimizing the amount of samples required to reach a satisfactory tracking performance. The agent having the least amount of samples required while reaching the same level of tracking performance will be the most sample efficient agent. Thus the focus will be on the samples required in a fixed time period and not the potential performance over an unlimited time period.

5.3. Results and discussion

A selection of four experiment setups were used to obtain these results, these experiment setups are a subset of 18 possible configurations with this mass-spring-damper system. The amount of masses, actions and tasks make up the set of configuration variables. After selecting these configuration variables, the experiment setup could use one of two adaptivity tests and one of the two tracking modes. The selection criteria for the variables of the experiment setup was focused around testing and proving the advantages of hierarchical reinforcement learning agents over non-hierarchical agents by expanding the state space while still resembling flight control.

The results from these experiments are summarized in table 5.6, table 5.7, table 5.8 and table 5.9 from section 5.3.2. These results center around two key aspects, tracking performance consistency and recovery consistency that show a perspective of how sample efficient and adaptive the agents are. From these tables, the most promising controller setups for the continuation of this research were selected for analysis of tracking and recovery performance though their time traces, the results can be found in fig. 5.10, fig. 5.11, fig. 5.12 and fig. 5.13 from section 5.3.6. to have a complete picture and grasp of the data presented in the table, a selection of the learning curves are presented here, additional learning curves and times traces are provided in appendix C. The learning curves give a better understanding about the adaptivity and sample efficiency of these agents but omit an understanding of tracking performance, thus the results presented in these three sections section 5.3.3, section 5.3.4 and section 5.3.6 should provide a more holistic view of tracking performance, adaptivity and sample efficiency. For these three aspects the results of each agent in the aforementioned sections, will be compared along three axes: experiment setup, adaptivity tests and tracking mode.

5.3.1. Hyperparameter Optimization

After an thorough initial analysis the following additional observations were made. The PPOC agents were not able to be more sample efficient than PPO using the TPE hyperparameter search and this experiment setup.

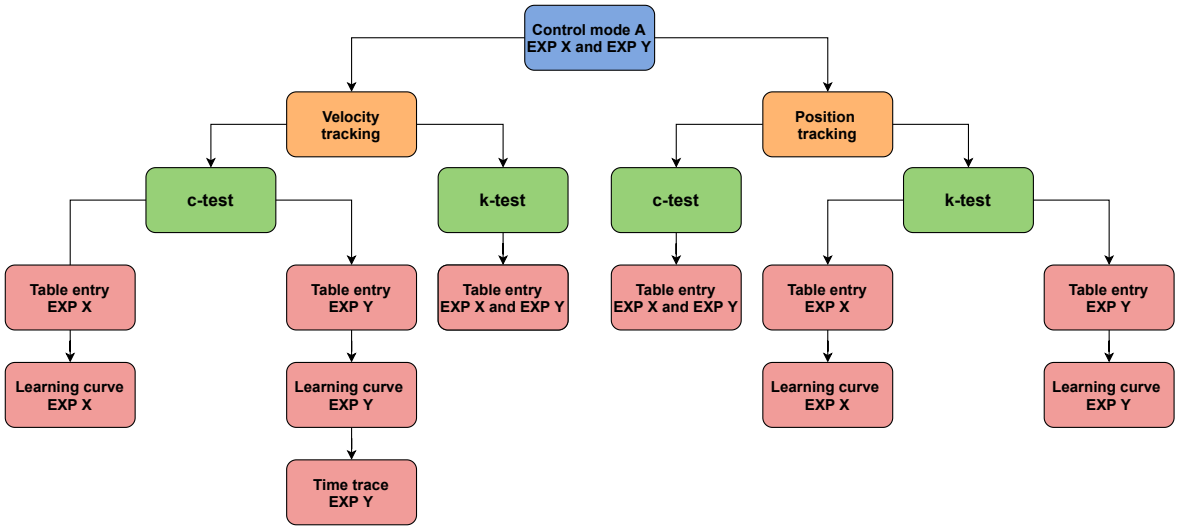

Figure 5.2: Built-up of results and comparisons

In fig. 5.3, the PPO agent used the optimal hyperparameters of PPOC-2 for an experiment using MSD-2. From this figure, it can be seen that the PPO agent is still more sample efficient than the most sample efficient PPOC agent while using hyperparameters that were not optimized for PPO. Though PPO is not stable throughout the whole run. A more fundamental explanation about PPOC being less sample efficient can be read in chapter 6.

The effects of hyperparameter tuning on the Option-Critic architecture can have considerable effects on learning performance. The same set of hyperparameters will result in similar adaptivity behavior over the different experiments, though the same hyperparameters cannot provide the same sample efficiency performance for different experiments before the activation of the adaptivity test for the PPOC agents, see fig. 5.4 and fig. 5.5. Where the agents are displaying consistent and distinctive indication of their sample efficiency. From these figures it can be concluded that hyperparameter tuning provides better agent behavior in terms of sample efficiency and adaptivity. But tuning for hyperparameters will not give more insight into how the agent will perform better than its non-hierarchical counterpart.

In order to enable higher sample efficiency for all PPOC agents, the TPE optimization should be repeated for all agents for all experiment setups. Given the fact that hyperparameter tuning is time-consuming and requires considerable computational resources to find optimal hyperparameters that are general enough while still retaining high sample efficiency and adaptivity to various circumstances. In the case of this research the various circumstances are the different experiment setups, tracking and control modes. And taking into account these factors and the information from the initial plots in fig. 5.4 and fig. 5.5. Then the effort required to perform hyperparameter tuning for each of these cases, should be taken into good consideration if it is worth the time and resources.

In addition the focus of this research is on analyzing specific properties of the hierarchical component of the Option-Critic architecture compared to its counterpart that is not using this component. So this means the benefits of using options should be analyzed. The amount of options is also a hyperparameter and to analyze the effect of this specific hyperparameter and property, the other hyperparameters should remain constant for a fair comparison between these different agents. Thus hyperparameter tuning for all hyperparameters does not provide extra relevant information in terms of answering the sub research question that started this preliminary analysis. Therefore for the continuation of the presentation of the results, the agents will all have non-optimal hyperparameters implemented unless when explicitly is stated that optimal hyperparameters were used.

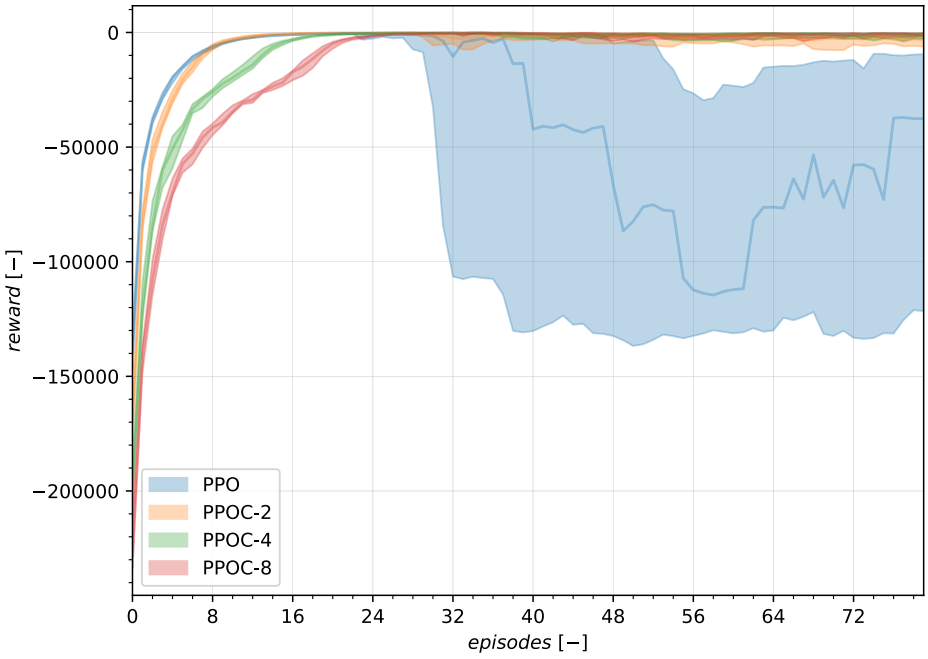

Figure 5.3: Hyperparameter Analysis for a MSD-2 with two tasks. Each line represents the interquartile range taken over 20 independent runs with different seeds.

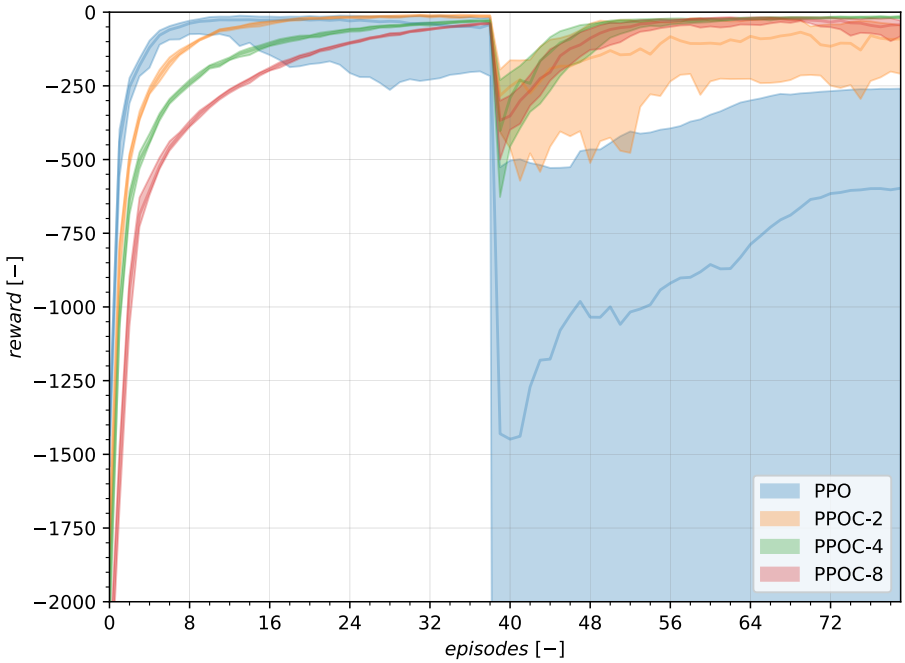

Figure 5.4: Results of velocity tracking in experiment 3 with the spring constant adaptivity test. The PPOC agents had their hyperparameters tuned with TPE optimization process. The hyperparameters of PPO were taken from the implementation of stable baselines [26] and was tuned for general optimal behavior.

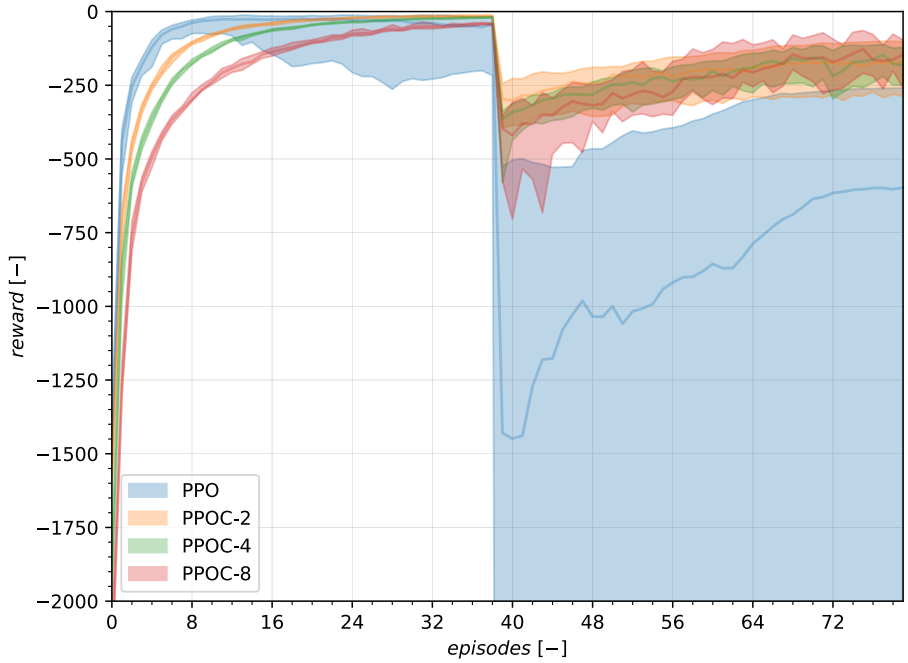

Figure 5.5: Results of velocity tracking in experiment 3 with the spring constant adaptivity test. The PPOC agents were not tuned. The same parameters as in fig. 5.4 were taken for PPO.

5.3.2. Overview of all experiments

An overview of all the runs are presented in this section, the analysis of these results are presented in section 5.3.3 and section 5.3.4. In those sections the agents are compared amongst experiment setup, adaptivity test and tracking mode. Note that the entries for the nominal case in each table are not matching exactly for the tables of velocity and position tracking. This discrepancy is inherent to the stochasticity of these agents. For velocity tracking a total of 800 runs were performed. These runs were split up in runs with c-test and k-test in table 5.6 and table 5.7, respectively. For position tracking a total of 800 runs were performed. These runs were split up in runs with c-test and k-test in table 5.8 and table 5.9, respectively.

Table 5.6: An overview of the consistency tracking rates for the **damping** constant adaptivity test while doing **velocity** tracking.

EXP	PPO		PPOC-2		PPOC-4		PPOC-8	
	nominal	recovery	nominal	recovery	nominal	recovery	nominal	recovery
1	12/18	00/18	19/20	13/20	20/20	20/20	19/20	17/20
2	14/18	00/18	20/20	02/20	15/19	04/19	16/20	02/20
3	11/20	00/20	20/20	04/20	20/20	19/20	17/20	18/20
4	12/20	09/20	16/20	08/20	15/20	06/20	13/20	07/20

Table 5.7: An overview of the consistency rates for the **spring** constant adaptivity test while doing **velocity** tracking.

EXP	PPO		PPOC-2		PPOC-4		PPOC-8	
	nominal	recovery	nominal	recovery	nominal	recovery	nominal	recovery
1	15/19	00/19	19/20	00/20	20/20	00/20	18/19	00/19
2	07/07	00/07	16/16	00/16	08/08	00/08	14/14	00/14
3	11/20	00/20	20/20	03/20	20/20	00/20	17/20	01/20
4	11/17	00/17	16/20	01/20	14/19	00/19	13/20	00/20

Table 5.8: An overview of the consistency tracking rates for the **damping** constant adaptivity test while doing **position** tracking.

EXP	PPO		PPOC-2		PPOC-4		PPOC-8	
	nominal	recovery	nominal	recovery	nominal	recovery	nominal	recovery
1	20/20	20/20	18/19	04/19	16/20	00/20	19/20	01/20
2	00/20	01/20	04/20	02/20	04/20	00/20	01/20	00/20
3	12/20	14/20	16/20	19/20	17/20	18/20	13/20	18/20
4	01/20	00/20	00/20	00/20	00/20	00/20	00/20	00/20

Table 5.9: An overview of the consistency tracking rates for the **spring** constant adaptivity test while doing **position** tracking.

EXP	PPO		PPOC-2		PPOC-4		PPOC-8	
	nominal	recovery	nominal	recovery	nominal	recovery	nominal	recovery
1	16/17	17/17	18/19	19/19	15/18	18/18	19/20	20/20
2	07/20	08/20	04/20	06/20	04/20	07/20	01/20	03/20
3	13/17	17/17	16/20	19/20	17/20	17/20	13/20	17/20
4	01/20	01/20	00/20	00/20	00/20	01/20	00/20	00/20

5.3.3. Longitudinal control: EXP1 and EXP3

Here the longitudinal control equivalence will be analyzed for a simple system (EXP1) and a more complex system (EXP3). For these systems an analysis on adaptivity and sample efficiency is given. For the adaptivity characteristics the velocity and position tracking will be analyzed separately using the c-test and k-test.

Adaptive velocity tracking

Starting with the c-test in table 5.6, for EXP1 it can be seen that PPOC-4 and PPOC-8 have much better recovery performance than for PPO. The options show their value in this case, where a non-hierarchical methods is not able to recover after the test. For EXP3 in the same table, a similar observation can be made. For these two rows from table 5.6 a trend can be identified. The learned options provide an advantage for tracking one velocity state in an unstable system. For both EXP1 and EXP3, PPOC-4 was performing the best, suggesting that the number of options should equal to 4 in order to have the best adaptive tracking performance for longitudinal control. This also means that 2 options were too few and 8 options were too much. The same observation can be made from the learning curves for EXP1 and EXP3 in fig. 5.14 and fig. 5.16, respectively. From these an initial option setting is determined, and for better performance a control designer can apply an hyperparameter optimization process to tailor for a specific application.

Continuing to the k-test in table 5.7 for EXP1 and EXP3 it can be observed that the agents all fail to recover. This observation suggests that adaptive velocity tracking is not possible for unstable spring constants. This can be explained by the fact that the spring constant is coupled to position states, thus when not controlling for position states the agent can only control indirectly for the position state. The results that are summarized in table 5.7 suggest that the agents have a hard time to learn this indirect control behavior of the dynamic system with instabilities introduced by the spring constant. The learning curves in fig. C.7 confirms this. The underlying phenomenon behind this indirect control behavior keeps recurring throughout the experiments and is explained in section 5.3.5.

Adaptive position tracking

Again starting with the c-test, generally seen the PPOC agents do not recover for EXP1 in table 5.8, while for PPO all runs recover from the damping constant instabilities. Interestingly this is exactly the opposite of

table 5.6 on how PPO and PPOC react to the same adaptivity test but with a different tracking mode for EXP1. The only explanation for this behavior is that a property of the Option-Critic architecture creates a consistent behavior that is the opposite of PPO in EXP1. This property being the switching between – and learning – multiple option policies versus learning a single policy with no switching required. The positive eigenvalue for the c-test for MSD-1 is quite large, see table 5.3, and this might result that the PPOC agents have a harder time to learn to control this large instability as it is switching between option policies, while PPO is learning a single policy.

For EXP3 in table 5.8 the PPOC and PPO agent's tracking performance rate are even increasing after the adaptivity test, indicating that the agents need more samples to learn position tracking. In this setup the PPOC agents are performing significantly better than PPO both before and after the test. Again the only difference being the options from the Option-Critic architecture. The observations made for EXP1 and EXP3 in table 5.8 suggests that the Option-Critic performs better in higher-dimensional state spaces. When taking into account the observations that were made in table 5.6 and table 5.7 for EXP3, then it can be confirmed that the Option-Critic architecture provides a RL agent with the capabilities to learn adaptive control for higher dimensional state spaces. An explanation for this might be that for higher dimensional state spaces, the same instability results in a lower positive eigenvalue (see table 5.5)than the eigenvalues for EXP1 in table 5.3. The lower positive eigenvalue and the fact that EXP3 only has one task, makes the setup of EXP3 the easiest experiment setup to learn for an Option-Critic. An additional factor that may contribute to the explanation of this finding is that the order of the c-test is lower than the order of the tracking state. The latter might allow for an easier transition in learning as the controlled state has a higher order than the instability that is introduced.

Continuing to the k-test for EXP1 and EXP3 in table 5.9 there is no significant difference between EXP1 and EXP3. Similar behavior for second order dynamics are observed for EXP1 and EXP3 where all agents needs more samples to learn position tracking. No differences are observed for EXP3 for the c-test and k-test, though for EXP1 there are differences to be observed with PPOC, where PPOC is not able to do recover in table 5.8 as was explained earlier.

Sample efficiency

An agent's in this research setup has two sample efficiencies that can be obtained from one learning curve. The first sample efficiency is derived from the nominal learning conditions which are the first 38 episodes. The sample efficiency is equal to the slope of the agent's learning curve, this is determined qualitatively. The steeper the learning curve, the more sample efficient the agent is. After the first 38 episodes, an adaptivity test is activated. The second sample efficiency is determined from the point that the agent stars learning a new behavior. Also in this case the agent's sample efficiency is determined by the slope of its learning curve. For the sample efficiency analysis, the position tracking with c-test and velocity tracking with k-test were excluded, as the recovery rates are low in table 5.7 and table 5.8

Starting with velocity tracking with the c-test in fig. 5.6, the PPO agent is clearly the most sample efficient before the adaptivity tests for both experiments. After the adaptivity test, PPOC becomes the most sample efficient for both experiments. Interesting to see is that the sample efficiency of all agents in the nominal phase are not affected by the larger state space, as the agents learning curve in the nominal phase for EXP1 and EXP3 are similar for each agent. The effect of the larger state space is noticeable during the recovery phase, both agents have a larger drop for EXP3. The effect is mainly attributed to the larger state space and not to the introduced instability, as the positive eigenvalues for EXP1 are larger than EXP3 as can be seen in table 5.3 and table 5.5, respectively.

An interesting observation, PPO recovers better for EXP3 than for EXP1, but both are not able to learn an good policy after the adaptivity test as was indicated in table 5.6. This effect after the drop can be attributed to the difference in magnitude of the positive eigenvalues, where EXP3 has the smallest magnitude resulting in policies that can obtain a higher reward than for EXP1.

Continuing with position tracking with the k-test in fig. 5.7, the PPO agent is most sample efficient for EXP1, but it becomes less evident for EXP3. For EXP3, the sample efficiency between PPO and PPOC become comparable. The learning response of both agents are fundamentally different from the velocity tracking setup. All agent's nominal behavior becomes less consistent, when going from EXP1 to EXP3. The inconsistency between runs make the slope of these learning curves less identifiable and thus no intelligible findings about sample efficiency can be made. The drop that was seen in fig. 5.6 is not as distinct here. The only drop that can be observed is for PPOC in EXP3. An explanation for this is that the agents are learning to control a state with slower dynamics – position tracking – thus might result in a larger 'learning inertia'. This learning

response is seen in all position tracking learning curves for agents that are able to learn a good policy. These learning curves can be seen in fig. C.13, fig. C.14, fig. C.15 and fig. C.16. Also a factor that might contribute to the similar learning behavior after the nominal phase, is the magnitude of the positive eigenvalues. In table 5.3 and table 5.5 it can be observed that the values are in the same order of magnitude. More research is needed to find the root cause of this learning response. If it is explained by the tracking mode, then this provides opportunities to design a more robust flight controller.

The learning curve fig. 5.16 shows that initially the agents are sample efficient when the number of options decrease. After the c-test, the agents sample efficiency behavior is the inverse of the initial observation. The observed behavior can be attributed to two possible explanations. The first being that this behavior is typical for Option-Critics. The second being that this behavior is the consequence of policies that have converged more than other policies might result in less flexible policies. Taking PPOC-8 as an example, then it can be seen that this agent has not reached its converged reward value as it is still learning, as a result it is more flexible and able to learn a new policy with fewer samples.

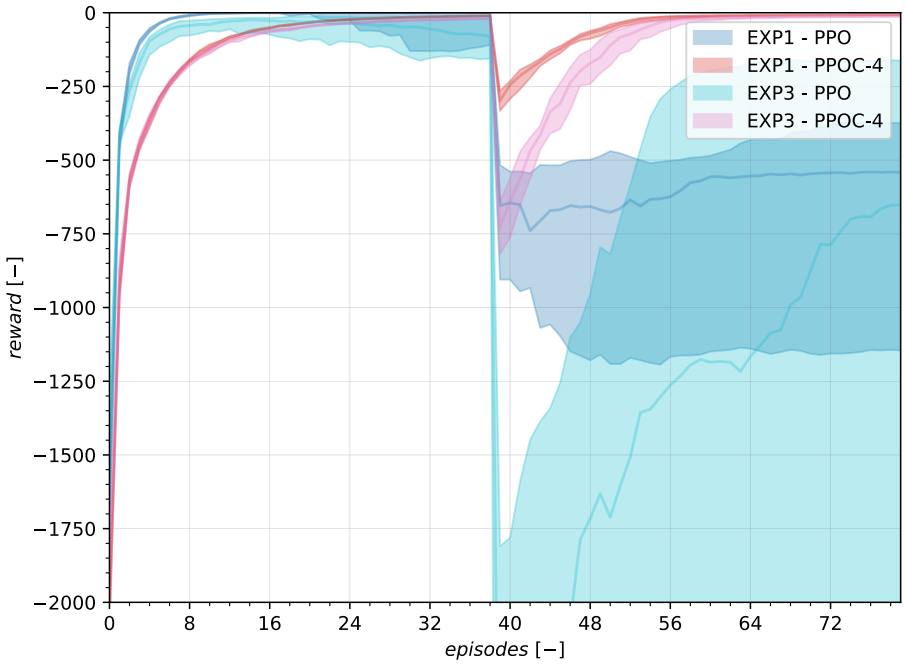

Figure 5.6: EXP1 and EXP3, velocity tracking with c-test.

Synopsis for longitudinal control

Summarizing the noteworthy results for the set of longitudinal controllers, PPO was outperforming PPOC in EXP1 with the c-test and position tracking. The PPOC agents provide for adaptive velocity control for higher-dimensional state space with damping constant instability, whereas the PPO agent under performs in this regard. Thus the focus should be on longitudinal body rate flight control when using PPOC and the advantages of HRL. In addition, the PPOC agents provide for better longitudinal position control than PPO for the c-test, though this might require many samples to learn this control behavior.

For complex longitudinal control all agents display good recovery behavior for both adaptivity tests during position tracking, see EXP3 in table 5.9 and table 5.8. Where it is interesting to see that the agents have a higher recovery rate than performance rate. This observation might indicate that the agents are less affected by the adaptivity tests during longitudinal position control, but need more samples than for longitudinal velocity control to learn a good policy. When looking at the learning curves fig. C.15 and fig. C.11 this is confirmed.

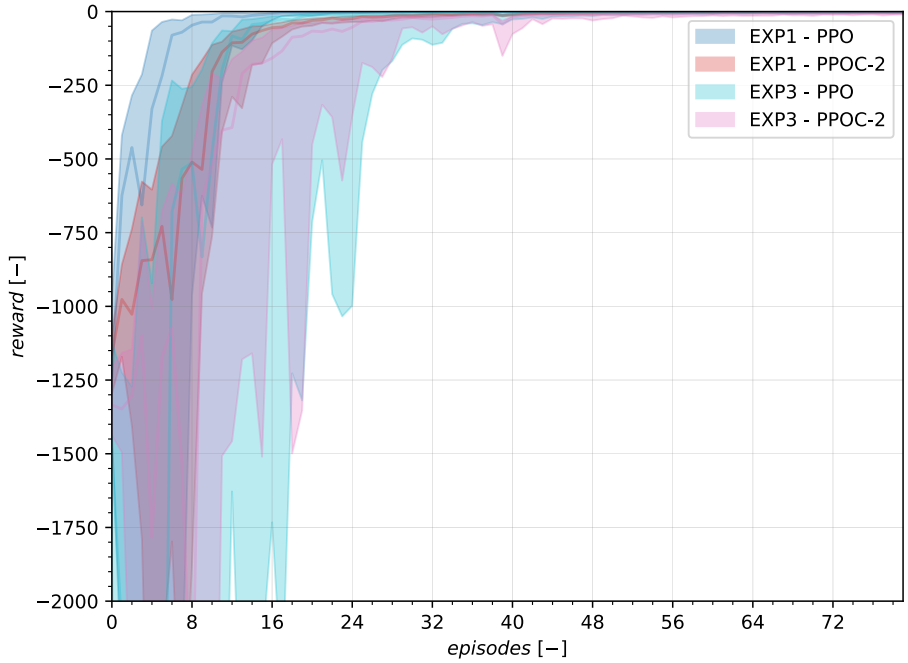

Figure 5.7: EXP1 and EXP3, position tracking with k-test.

5.3.4. Lateral control: EXP2 and EXP4

Here the lateral control equivalence will be analyzed for a simple system (EXP2) and a more complex system (EXP4). For these systems an analysis on adaptivity and sample efficiency is given. For the adaptivity characteristics the velocity and position tracking will be analyzed separately using the c-test and k-test.

Adaptive velocity tracking

Starting with the c-test and observing the differences between agent behavior for EXP2 and EXP4 in table 5.6, it can be seen that for EXP4 there is better recovery behavior for all agents. This shows that all agents have better adaptive behavior for more complex environments with lateral control. The same observation was made for adaptive behavior complex longitudinal control and explained in section 5.3.3. When comparing amongst agents for EXP2 and EXP4 in table 5.6 then it can be seen that PPOC-2 is the most distinctive and best performing agent regarding tracking and recovery. The options might provide an extra advantage over PPO, but from this table that is not completely evident. A closer inspection is needed to observe the advantages of options. Luckily the learning curves in fig. 5.15 and fig. 5.17 provide a more evident distinction, there all PPOC agents are more – or as – consistent and attain a higher reward than PPO for both learning curves.

Continuing to the k-test and looking at EXP2 and EXP4 in table 5.7, then no additional insights are gained regarding the benefits of using options. The only observation that is notable is that generally seen all agents were not able to recover from both experiments. In fig. C.8 it can seen be that PPOC is more consistent and provide a slightly higher reward than PPO, the reason for the agent not being able to learn for these experiment is explained in section 5.3.5.

Adaptive position tracking

For the c-test for EXP2 and EXP4 in table 5.8 the agents were generally not able to execute position tracking and also not able to recover from instabilities. Indicating that multitasks for second order dynamics are hard to learn for both hierarchical and non-hierarchical agents. When comparing the experiment sets EXP1 with EXP2 and EXP3 with EXP4 in table 5.6, table 5.7, table 5.8, and table 5.9 it can be seen that in all cases that the lateral control experiments EXP2 and EXP3 are having lower performance tracking rates and recovery rates when compare to the longitudinal control experiments EXP1 and EXP3.

For the k-test for EXP2 and EXP4 in table 5.9, all the agents are able to learn a good policy for EXP2 but not for EXP4. On closer inspection of the learning curves of EXP4 in fig. C.16, it can be seen that all agents obtain very low rewards when looking at the scale of the y-axis. Though a clear distinction can be made between the the learning behavior of PPOC and PPO, where PPOC is showing significant better learning behavior with increasing options.

Sample efficiency

Starting with velocity tracking with the c-test in fig. 5.9, the PPO agent is clearly the most sample efficient before the adaptivity tests for both experiments. After the adaptivity test, PPOC becomes the most sample efficient for both experiments. For longitudinal control it was seen that sample efficiency of all agents in the nominal phase are not affected by the larger state space, though here the sample efficiency is lower for the larger state space.

The effect of the larger state space is the opposite for longitudinal control during the recovery phase, both agents have a smaller drop for EXP4 than for EXP2. The effect might be attributed to the larger state space and the introduced instability that result in a lower eigenvalue for EXP4 or the multi-task setup. More research is needed to find the root cause of this learning response.

An interesting observation, PPO and PPOC are not able to learn an good policy after the adaptivity test as was indicated in table 5.6. This effect can be attributed to the difference in magnitude of the positive eigenvalues, where EXP4 has the smallest magnitude resulting in policies that can obtain a higher reward than for EXP2.

Continuing with position tracking with the k-test in fig. 5.7, the PPO agent is most sample efficient for EXP2. The same analysis from longitudinal control can be applied here. Only there the agents have a harder time to learn a good policy as the position tracking is a second order dynamic as explained in section 5.3.5. Given that position tracking is a second order dynamic, EXP4 has a larger state space and two tasks, then that might explain the learning behavior of PPO and PPOC observed for EXP4 in fig. 5.9. Evidence that position tracking for two tracking tasks is harder to learn, can be observed by the larger y-axis scale for fig. 5.9 than for fig. 5.7.

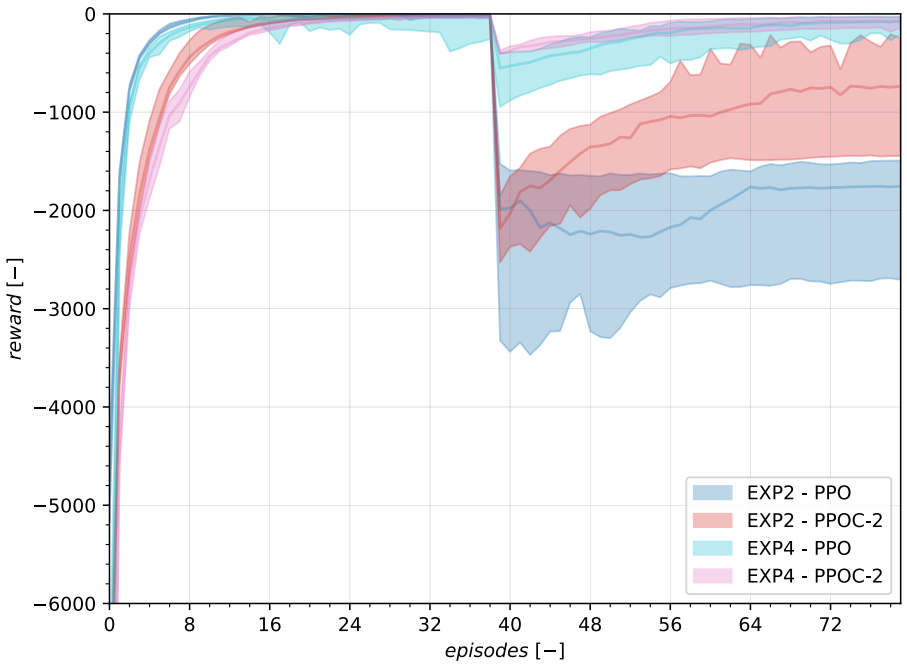

Figure 5.8: EXP2 and EXP4, velocity tracking with c-test.

Note that also the adaptive sample efficiency behavior that was observed in fig. 5.16 can also be observed in fig. 5.17, where the order of the most sample efficient agents before the test is reversed after the test. These

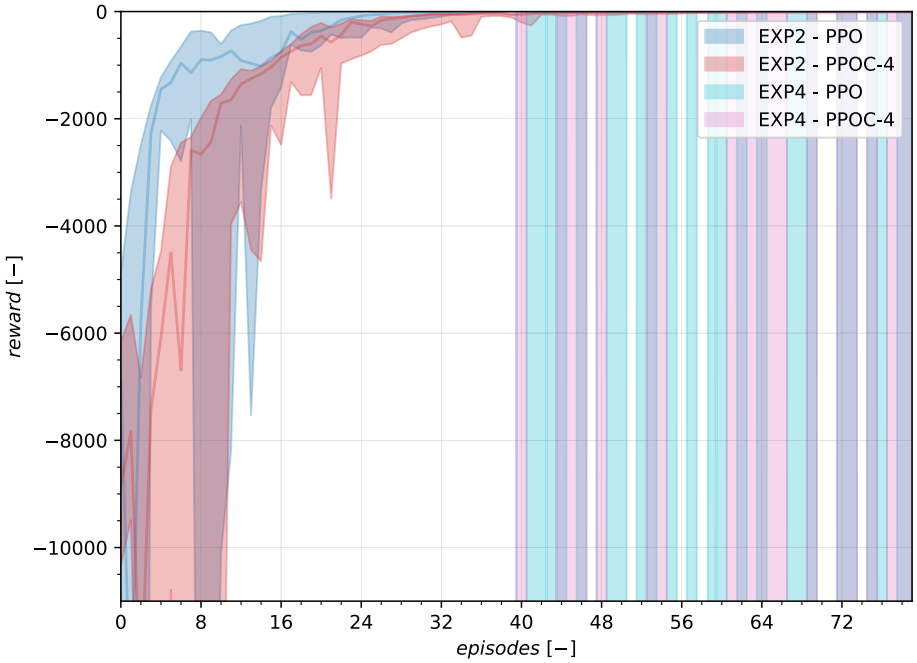

Figure 5.9: EXP2 and EXP4, position tracking with k-test.

observations support the hypothesis that the sample efficiency after the test is mainly influenced by the level of convergence. In order to have better evidence a test with PPO should be performed that focus on activating an adaptivity test before PPO has reached a certain level of convergence.

So an interesting application of this information is to train agents not to its fullest performance but let them remain sub-optimal to have the best adaptivity behavior aka exploration and exploitation vs exploration. In the continuation of this research extra tests and analysis should be conducted to asses if this behavior stems from level of convergence or from the Option-Critic.

Synopsis lateral control

The Option-Critic agents provide a higher consistency and higher reward than for PPO. When comparing the lateral control results with the longitudinal control results, then it can be concluded that providing agents with multiple tasks make learning a good adaptive policy a harder exercise for all agents. But the Option-Critic provides a framework that performs considerably better than an agent without this framework and is able to perform adaptive lateral control for unstable damping while tracking velocity states.

It should be noted that also in the lateral experiments the same observations can be made about that the tracking control mode should equal its instability mode in order to learn a good tracking policy.

For both PPO and PPOC, it was not possible to learn an adaptive lateral controller for a large state space while tracking position states. This setup was proven to be the most difficult. This indicates that even though hierarchical methods are known to be more sample efficient for high-dimensional and complex environments. They are still not able to learn for the most difficult continuous control tasks in this setup.

5.3.5. n-order dynamics and n-order instability

From section 5.3.3 and section 5.3.4 a recurring phenomenon was observed, the agent's learning has a strong correlation between the adaptivity test and the tracking mode. The focus of this section will be on this phenomenon taking the longitudinal control experiments as case study. The findings of this case study applies also for the lateral control experiments.

In table 5.9 and table 5.8, the runs are generally seen not recovering from the adaptivity test. A general pattern was identified that the experiments with an adaptivity test that had an instability constant that was

not directly linked with the state that was being tracked, resulted in almost no to no recovery. The only exception being EXP3 in table 5.8, an explanation was found in section 5.3.3. When continuing on this general pattern, then this general pattern can be explained more in-depth by the physics of the environment model.

For example when taking the position control with the k-test, then the negative spring constant and the instability resulting from that, can be directly counteracted by position control, as the spring force is the spring constant multiplied by distance. Where the distance is the state that is being tracked. In the case of velocity tracking with the k-test, then a negative spring constant is introduced after the nominal phase, while the agent is tasked to control the velocity. The only way for the agent to interact with the environment and thus controlling this state is by exerting force. The agent somehow must develop a policy that exerts a force on a mass that that counteracts the spring force and tracks the velocity with the desired reference signal. So the agent must somehow counteract this instability while following a reference signal. This setup for an agent requires a more specific force input signal to be given to the environment than when the force component for tracking is the same the force component that counteracts the instability. This form of control will be called indirect control. The resultant unstable force that interacts with the rest of the system will be called unstable indirect dynamics.

The understanding gained from the physics of the model gave more insight into how the agent's task can be more difficult through unstable indirect dynamics and result in learning bad policies. So for the agents to learn good policies considerate trade-offs concerning indirect dynamics should be made to assure a higher probability of learning a good policy. An insight can be developed by the general pattern observed for bad policies and the experiment setup. The agents will learn a good policies when there are no unstable indirect dynamics. For this to happen the introduced dynamics by unstable constants need to align with the dynamics of the tracking mode. For example: an experiment setup with velocity tracking and a c-test will result in agents that is able to learn a good policy, see table 5.6 .

The output of an agent is a force on a mass. The temporal relation between force and velocity is in the equations of motion. There the force is equal to mass times acceleration. From basic physics, the temporal relation between acceleration and velocity is a single integration and between acceleration and position is a double integration. So given this fact, a few extra terms are introduced when building on the existing idea of indirect dynamics. A change in the damping constant in a MSD system will have direct effect on the damping force. The state related to the damping force is velocity. As seen from the agent's perspective the force it exerts on the system is one integration away to get to velocity. The dynamics that the agent needs to learn to control with velocity tracking will be called first order dynamics and the introduced damping instability will be called first order instability. In the same train of thought, the agent needs to learn second order dynamics when learning for position control and the introduced spring constant instability with the k-test will be called second-order instability.

So for an agent to have a higher probability to learn a good control policy the n-order dynamics should match the n-order instability introduced by the adaptivity test. In the design of a flight controller this should be taking into account. This means that it is not possible to train one agent to be a controller that is encompassing both spring and damping instabilities for this specific application of the algorithms.

5.3.6. Velocity tracking time traces for EXP3 and EXP4 with c-test

In this section only the time traces of velocity tracking for EXP3 and EXP4 with the c-test for one PPO and one PPOC agent are presented. These time traces were selected based on the highest tracking performance rate and highest recovery rate for EXP4, these rates can be found in table 5.6. The tracking performance rate is determined by the amount of runs that were able to have a mean reward over the last five episodes to be to equal -50 or higher. The recovery rate is the same as the tracking performance rate with only one difference, the tracking performance rate is calculated with data before an adaptivity test and the recovery rate is calculated with the data after the test. A complete overview of these rates for all runs are provided in table 5.6, table 5.7, table 5.8 and table 5.9. The reference tracking plots fig. 5.10, fig. 5.11, fig. 5.12 and fig. 5.13 are time traces were the IQR is taken over 20 runs. These time traces correspond to learning curves given in fig. 5.16 and fig. 5.17. For the reference tracking plots for EXP1 and EXP2 the reader is referred to appendix C.

In fig. 5.10 and fig. 5.11 the time traces of PPO and PPOC-4 are given, respectively. When looking at the left columns the agents learn both a good policy for velocity tracking and while having a large state space with one task. For the first three seconds, the PPO is less consistent than the PPOC-4 agent this level of consistency can also be observed in table 5.6. When looking at the right column of these plots, then the PPOC-4 agent is displaying similar and consistent tracking performance with slight oscillatory tracking that is observed in the error signal. In the case of PPO the tracking performance is not consistent and is overshooting in most runs.

For velocity tracking for EXP3 with the c-test, the PPOC-4 outperforms PPO, thus indicating that the Option-Critic agents are better suited for adaptive control in large state spaces than for agents without an Option-Critic framework. This observation is seen in fig. 5.16 where the number of options shows a linear trend in sample efficient adaptive behavior. The more options, the more sample efficient the adaptive behavior will be.

In fig. 5.12 and fig. 5.13 the time traces of PPO and PPOC-2 are given, respectively. Two tracking tasks are seen with their respective error signals. The agents are quite comparable in consistency and tracking performance, though it was expected that the consistency would be higher for PPOC-2 before the test when looking at the rates 12/20 and 16/20 for PPO and PPOC-2 in EXP4 from table 5.6, respectively. PPOC-2 might still be more consistent than PPO, but this might be masked by the noisy tracking behavior of PPOC-2. When looking at the learning curves in fig. 5.17, then all agents converge to a point where reasonable policies were learned but did not satisfy the recovery rate requirement (reward should be -50 or higher), this might explain the discrepancy in expectation from table 5.6 and results from the time traces between PPO in fig. 5.12 and PPOC-2 in fig. 5.13. In addition, in both the left and right column in fig. 5.13 the PPOC agent shows noisy tracking behavior that is visible as the dark red line is thicker than the same colored line for PPO. This behavior is seen for all velocity tracking runs for PPOC, it can be explained by the high frequency control behavior required for velocity tracking that leads to high frequency option switching.

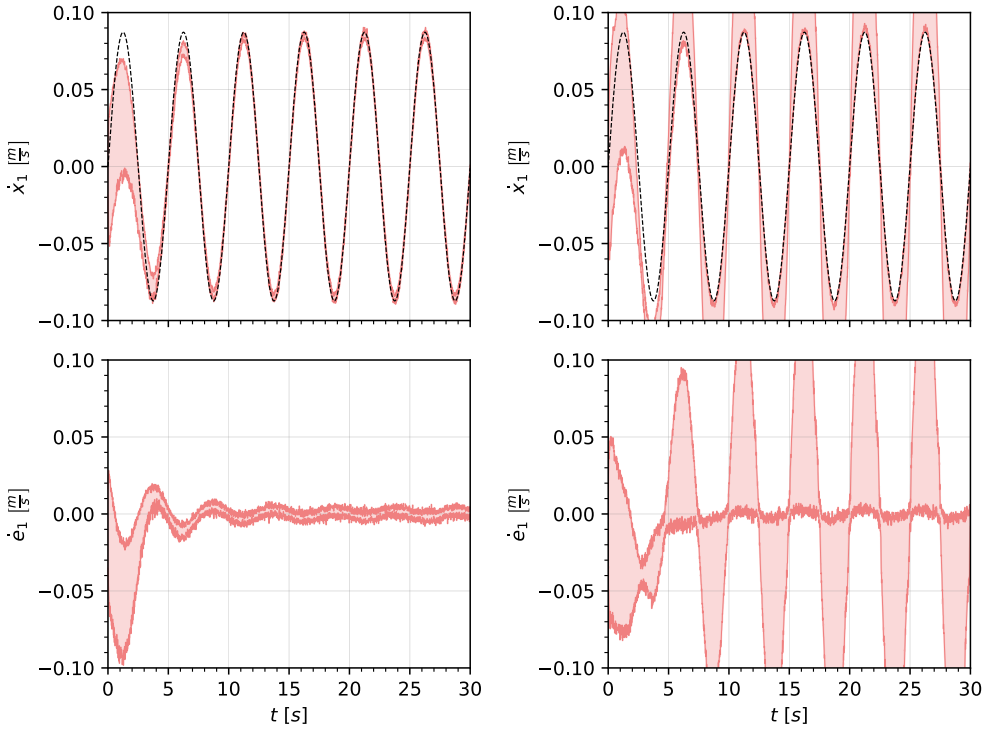

Figure 5.10: Results of velocity tracking in experiment 3 with the damping constant adaptivity test for PPO. In the left and right columns are the time traces before the test and after the test, respectively. See fig. 5.16 for the corresponding learning curves.

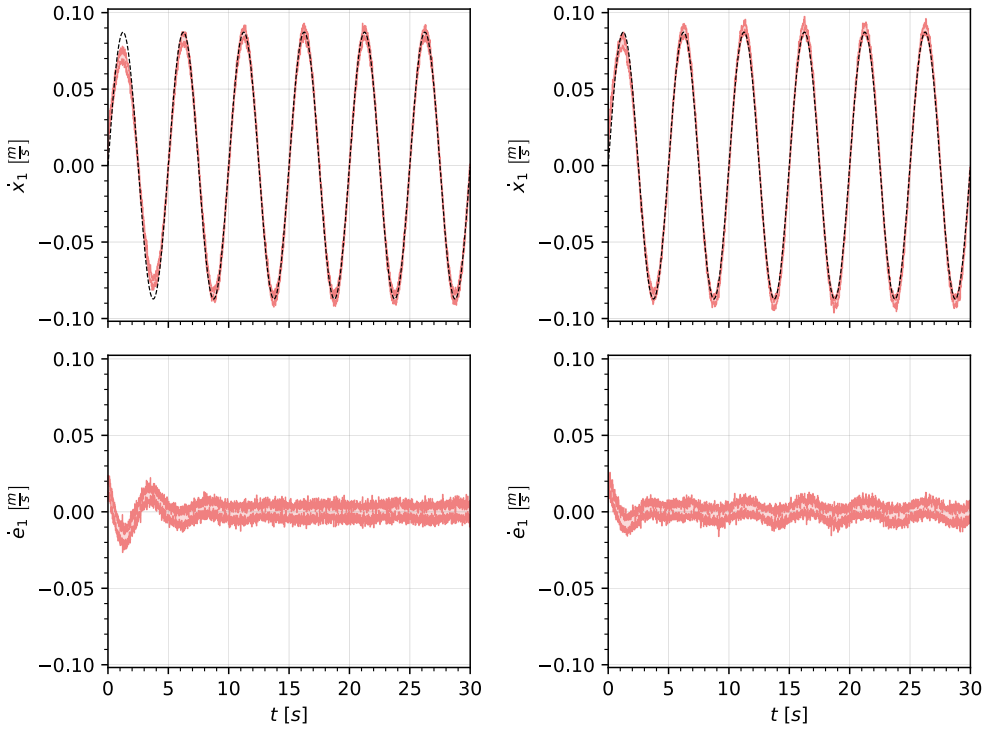

Figure 5.11: Results of velocity tracking in experiment 3 with the damping constant adaptivity test for PPOC-4. In the left and right columns are the time traces before the test and after the test, respectively. See fig. 5.16 for the corresponding learning curves.

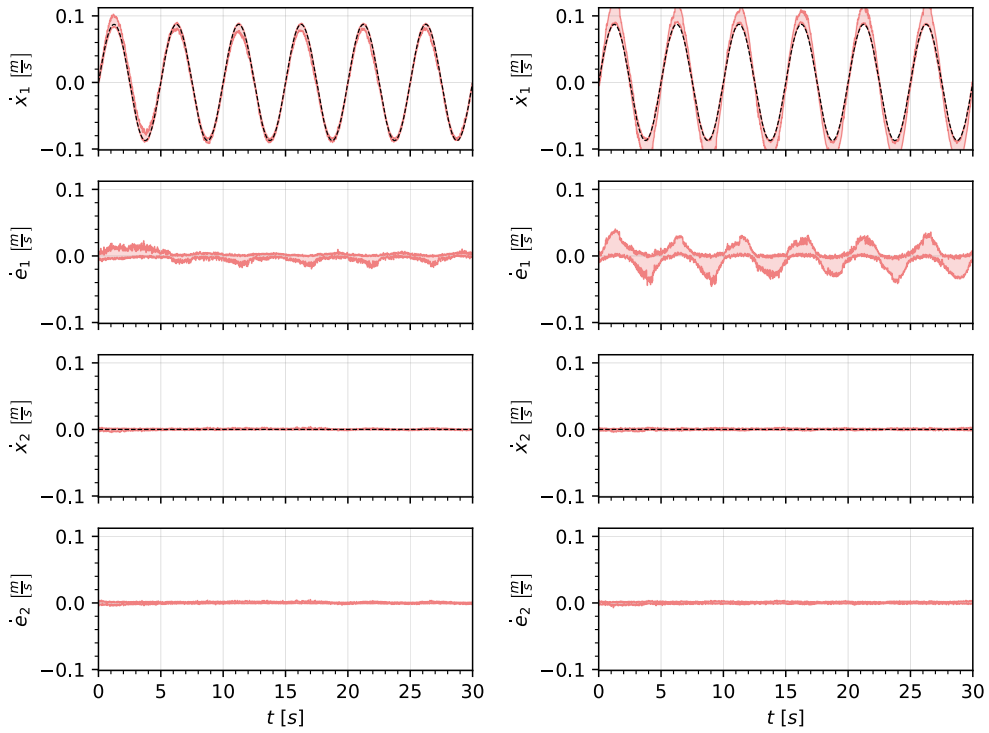

Figure 5.12: Results of velocity tracking in experiment 4 with the damping constant adaptivity test for PPO. In the left and right columns are the time traces before the test and after the test, respectively. See fig. 5.17 for the corresponding learning curves.

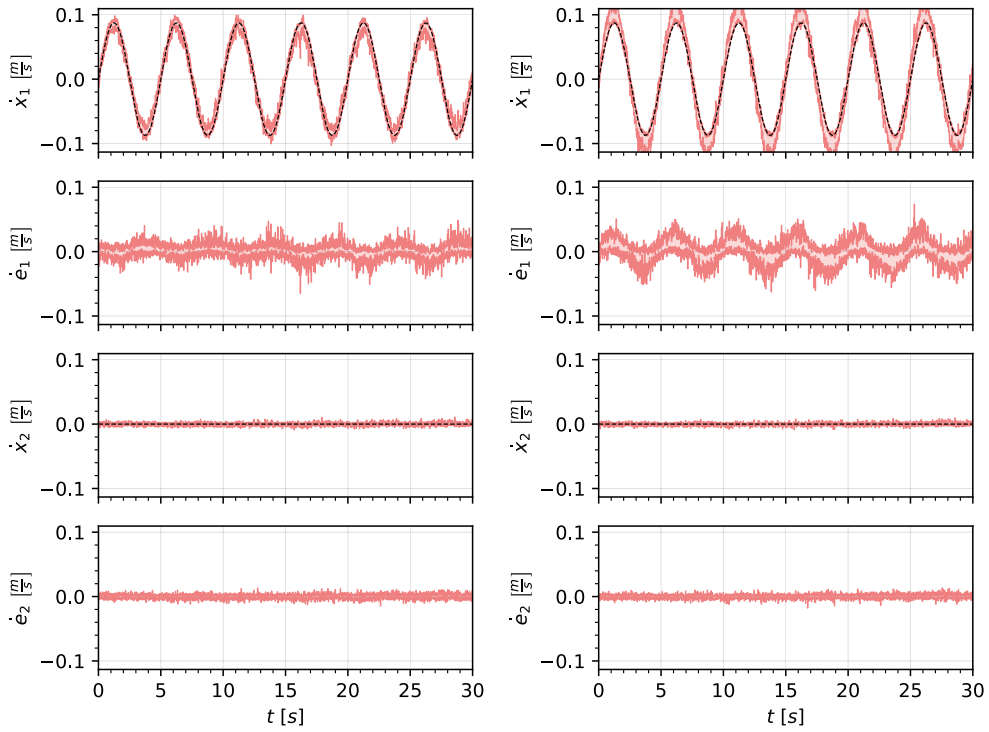

Figure 5.13: Results of velocity tracking in experiment 4 with the damping constant adaptivity test for PPOC-2. In the left and right columns are the time traces before the test and after the test, respectively. See fig. 5.17 for the corresponding learning curves.

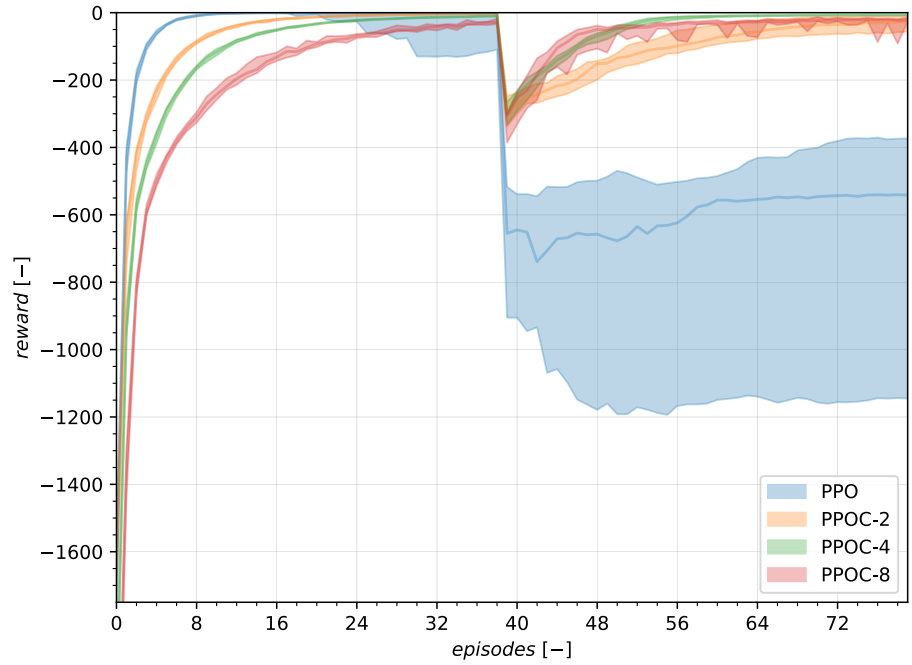

Figure 5.14: EXP1, velocity tracking with c-test.

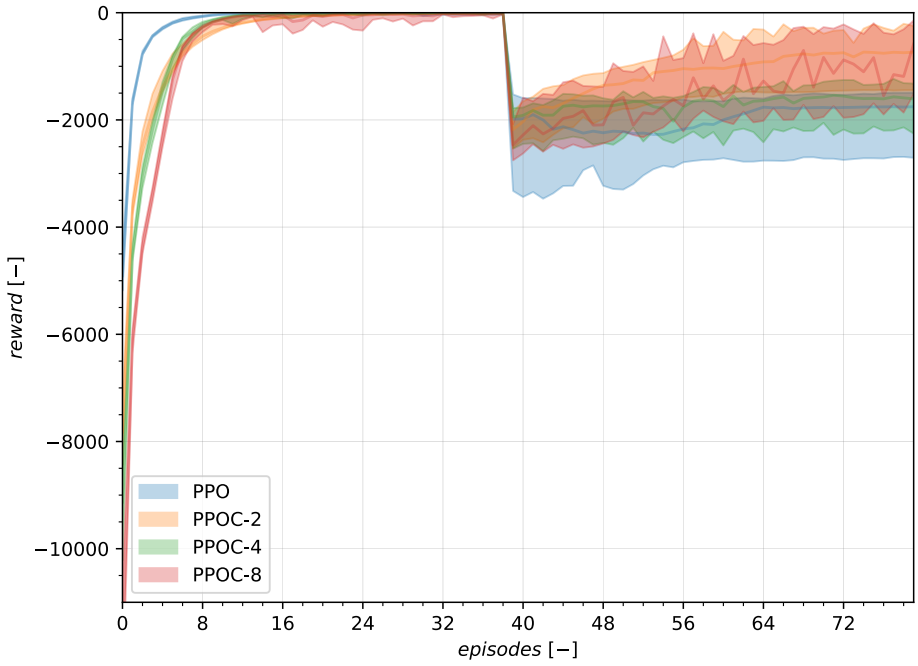

Figure 5.15: EXP2, velocity tracking with c-test.

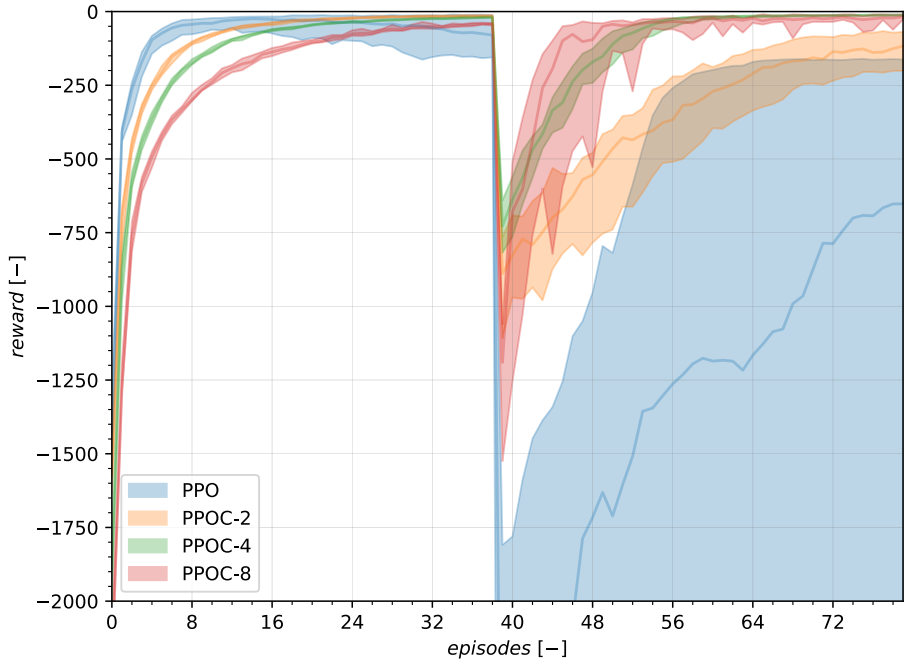

Figure 5.16: EXP3, velocity tracking with c-test.

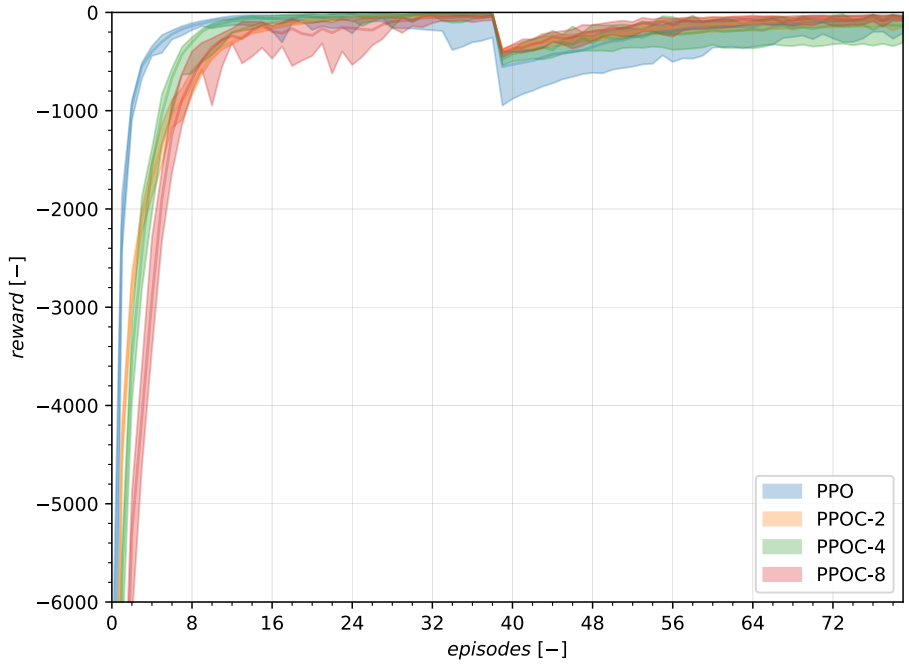

Figure 5.17: EXP4, velocity tracking with c-test.

5.4. Conclusion

The goal of this chapter is to provide an answer to the third research question: **What are the benefits of the proposed hierarchical policy gradient technique over the proposed non-hierarchical policy gradient technique on a mass-spring-damper system when exposed to unexpected changes?** This question is answered by answering its sub-questions that are restated here:

The Option-Critic has one hyperparameter that needs to be set, which is the number of Options n. In order to compare the benefits of Hierarchical Design and, in this case, in the form of Options, three Option-Critic settings – $n = 2$, $n = 4$, and $n = 8$ – have been tested against PPO without Option-Critic architecture. From initial tuning efforts, it is found that hyperparameter tuning for PPO and PPOC did not result in fair and interpretable comparison between non-hierarchical and hierarchical methods as it might obscure the fundamental differences. Hyperparameter tuning is useful for a more detailed design phase as it can enhance sample efficiency and/or adaptivity. The methods are tested on a mass-spring-damper system that can extend its state and action space. The mass-spring-damper system allows for easy analysis of the agent's behavior and allows for versatility of experiment setup, which are desirable traits in the agent's algorithm development. The benefits in question are mainly focused on three aspects that are found to be the most interesting and relevant for flight control: adaptivity, sample efficiency and tracking performance.

a) ***What is the performance regarding adaptivity?***

The Option-Critic is superior here. Overall, the PPOC methods show a higher recovery consistency rate than for PPO. Thus PPOC is better concerning adapting to unexpected changes. An explanation is that the adaptive behavior of PPOC agents has a lower final reward during the nominal phase, whereas PPO has a higher final reward. As a result, the PPOC are less overfitted. As a consequence, the PPOC are more adaptive. The higher adaptivity results in the higher sample efficiency during the recovery phase with PPOC. The PPOC method with the most Options – PPOC-8 – is often the most sample efficient and having the highest final reward during the recovery phase. On the contrary, in the nominal phase, the PPOC-8 agent is the least sample efficient and has the lowest final reward. From this observation, a clear trend is identified that when more options are available that the PPOC agents are more adaptive and more sample efficient during the recovery phase than during the nominal training phase.

b) ***What is the performance regarding sample efficiency?***

In general, it is found that PPOC-2 was more sample efficient during recovery for lateral velocity control and PPOC-4 for longitudinal velocity control. The number of Options has different effects for longitudinal control than for lateral control. For longitudinal control, the sample efficiency in the nominal phase is with increasing Options, less sample efficient. On the other hand for lateral control, the number of Options has almost no effect on the nominal phase. In addition, it is observed that the PPOC agents always perform better in the recovery phase in terms of sample efficiency, but that the PPO agent is always the most sample efficient during the nominal training phase.

c) ***What is the performance regarding reference tracking?***

The tracking performance is shown for longitudinal and lateral velocity control with the c-test with large state space. From the results, it is found that PPO is more smooth for lateral control and underperforming for longitudinal control during the recovery phase. The tracking performance of PPOC is overall the best for both nominal and recovery phases and for both control modes. In general, PPOC has a smaller error signal for both control modes, but lateral control is less smooth and noisier due to the high frequency switching between options. As a result, it has a more aggressive form of control during velocity tracking.

III

Wrap up

6

Conclusions

Advancements made in reinforcement learning provides opportunities to the development of autonomous and intelligent flight control system for novel and conventional aircraft configurations. Control systems that utilize reinforcement learning can enable the ability to learn from scratch without detailed model information, contribute to low workload for pilots and ensure safe flight operation during unexpected changes to the environment. The need for a model-independent and adaptive flight controller is identified, where this need can already be satisfied with existing methods. The existing methods found in Approximate Dynamic Programming and Incremental Approximate Dynamic Programming are mainly applicable to small state spaces, whereas Actor-Critic Design methods found in Deep Reinforcement Learning provide opportunities to extend Actor-Critic Design methods for model-independent and adaptive control to large state spaces. Though problems with large state spaces often lead to low sample efficiency. A reinforcement learning method extended with Hierarchical Design can result in faster learning for high-dimensional state spaces by creating low-dimensional abstractions of the complete problem domain. The advantages of Actor-Critic Design and Hierarchical Design are combined and lead to the following research objective:

Contribute to the development of a novel model-independent and adaptive controller for a continuous, high-dimensional, partially observable, and stochastic problem domain by investigating a hierarchical policy gradient reinforcement learning flight controller for a fixed-wing aircraft that enables sample efficient flight recovery from unexpected changes to aircraft dynamics.

From the research objective, four research questions were formulated (in section 2.2) to answer the main research question and ultimately fulfill the research objective. The first three research questions will be restated here and answered on a high level. If desired to have a more extensive answer to the research questions that includes the sub-questions, then the reader is referred to section 3.4, section 4.5 and section 5.4. The first three research questions were answered in the preliminary analysis and are part of a preliminary report. The fourth research question is answered using the research and analysis given in part I.

1. **What are the requirements for a fixed-wing aircraft flight controller?**

An all-encompassing flight controller should require six-degree-of-freedom control and involves a high-dimensional state space with coupled dynamics. Designing a controller for such complexity from scratch is arduous. A step-by-step approach is needed to simplify and gain understanding when designing the controller. The complete six-degree-of-freedom model is decoupled into symmetric and asymmetric motion, for which it respectively needs a longitudinal and lateral controller. A reinforcement learning controller for model-independent and adaptive flight control should learn from scratch, a probabilistic mapping of states to control actions. This is followed by re-adjustments in the probabilistic mapping while doing this in a sample efficient and online fashion. In addition, when such a controller accounts for variation in time scale and magnitude of aircraft states, it might speed up learning.

2. **What are the current state-of-the-art reinforcement learning methods that are suitable for flight control?**

From the literature review in chapter 4, it is found that an Actor-Critic Design with Proximal Policy Optimization is able to provide for prediction and control for high-dimensional state space in an on-policy fashion. The method alternates between optimizing the Actor-network and sampling from the environment. The authors of the method have shown that when combined with a Generalized Advantage Estimation function that utilizes state value estimates from the Critic network, then this function will greatly increase sample efficiency by reducing variance in its value estimates at the cost of some bias.

A Hierarchical Design method most suitable for flight control that can extend non-hierarchical policy gradient methods with Actor-Critic Design is the Option-Critic architecture. The Option-Critic architecture has the ability to recover from sudden changes made in the environment and is able to create end-to-end temporal abstractions by learning intra-option policies and termination functions. The two main advantages provide an opportunity to cope with in-flight failures and can have the ability to distinct timescales of aircraft states, respectively.

The Actor-Critic Design with Proximal Policy Optimization extends with the Option-Critic architecture. The resulting method is called Proximal Policy Option-Critic (PPOC) and has a successful implementation for continuous action spaces in environments created with the MujoCo physics engine. The Actor-Critic design with Proximal Policy Optimization and Generalized Advantage Estimation function is the non-hierarchical policy gradient baseline. The extension of the non-hierarchical method with the Option-Critic architecture is the hierarchical policy gradient method. These methods are used for comparison in order to identify and verify the benefits of Hierarchical Design.

3. **What are the benefits of the proposed hierarchical policy gradient technique over the proposed non-hierarchical policy gradient technique on a mass-spring-damper system when exposed to unexpected changes?**

One of the two main benefits of the Option-Critic architecture that is identified in the literature review is confirmed. The Option-Critic adapts to unexpected changes to the environment, where increasing the Options results in a higher sample efficiency during the recovery phase. The method is not more sample efficient than its non-hierarchical counterpart during the nominal learning phase. A possible explanation is that the Option-Critic needs to learn multiple intra-option policies and termination functions. These policies are each equal to a policy that is learned by a Proximal Policy Optimization method. As a result, the Option-Critic architecture has fewer samples for each intra-option policy, making the algorithm less sample efficient.

Finally, the tracking performance of the Option-Critic is found to be generally better than its non-hierarchical counterpart as the error signal is smaller, but it is noisier in cases when the error signals of both methods were at comparable magnitude.

The differences between the Option-Critic and its non-hierarchical counterpart alludes to holding Options for a longer time period might improve tracking performance and sample efficiency during the nominal phase as the Option-Critic with only one Option should exhibit the same performance as its non-hierarchical counterpart.

4. **How should the proposed hierarchical policy gradient method be implemented for flight control of a fixed-wing aircraft?**

a) What is the performance regarding adaptivity, sample efficiency, and reference tracking when exposed to unexpected changes?

Surprisingly the anticipated sample efficiency gain through hierarchical reinforcement learning is not realized for the offline training for both the mass-spring-damper system and aircraft model. PPOC using a single Option and PPO – a single policy method – are the most sample efficient, and the sample efficiency of learned Options decreases with increasing Options. This is explained by the fact that each additional learned Option requires more parameters to be learned. An additional explanation might be in the fundamental difference in the classical application of Options, where hand-crafted intra-option policies have prior knowledge embedded in them. Embedding prior knowledge, in general, allows reinforcement learning methods to be more sample efficient.

Summarizing the results with regards to adaptivity, PPOC with multiple learned Options give higher success rates than PPO for offline adaptivity. In addition, PPOC with multiple learned Options leads to higher success rates than PPOC-1 during tracking of a height profile while having a structural failure of the horizontal tailplane, sign change of pitch damping, and generalizes to a different aircraft. PPOC is able to extend its learning to a different aircraft without requiring extra offline training. This shows that having more learned Options result in the ability to generalize over different aircraft. In addition, it allows for a stabilizing effect during online learning and adaptivity. Thus multiple learned Options allow for more complex control behavior and be adaptive in a reliable manner.

In regards to reference tracking, the PPOC agent is able to follow the reference signals for both MSD and aircraft systems in the offline and online setting. Though, PPOC is not displaying the desired actuation behavior even after applying the strategy for integrating the actions. Two sources that contribute to the noisy output of the PPOC agent are identified. The first source is the interaction between the termination function and policy over options causes high frequency switching between Options. As a consequence, PPOC learns a high-frequency control policy. The second source is the underlying stochastic policy method. This is seen for the PPOC-1 agent, where the high-frequency output is the result of the Gaussian distribution used for sampling the actions.

b) How can sample efficiency be improved?

From the time traces of the Actor and Critic MLP weights, the changes made during online learning for both networks are similar. This indicates that the networks are responding similarly to small changes in the environment. When allowing the Actor and Critic networks to have a shared layer, then this reduces the amount of sample needed to adapt to a different environment and thus improving the sample efficiency. The implementation of this sharing is left for further research.

c) How can reference tracking be improved?

PPOC is able to track a height reference with high accuracy, though the high-frequent output of the agent is undesirable for flight control signal as it shortens the durability of the actuator and introduces unwanted vibrations. The hierarchical method's high-frequency control is attributed to switching between intra-option policies. This behavior is apparent in the action signal during reference tracking. The high frequency and high gain control initially observed for the mass-spring-damper system were reduced by handling the agent's output as a differential and by limiting the increment per time step. As a result, the reference tracking performance is improved, the high gain control has disappeared, and the high-frequency control is reduced. Though, the high-frequency control remains and additional techniques are needed to reduce the high-frequency control even further.

The observed high-frequency output can be reduced with the following three approaches. The first is a simple solution where the high-frequency output can be reduced by passing it through a low pass filter, though this will introduce lag and less control by the agent. Another more fundamental approach is to have more control over the stochasticity of the stochastic policies by making the entropy variable with a temperature variable. The last proposed approach is in the direction of having more control over the switching induced by the termination function. For the latter, there is an existing method that adds a deliberation cost [22] to the termination gradient. The deliberation cost lets the agent switch less frequently by incurring a cost for switching.

All in all, in answering the research objective. PPOC is capable of learning how to control an aircraft's inner loop dynamics, though it requires offline training before it can be applied in an online setting. Still, the method does not require any model information and solely uses samples resulting from interactions with the model. Multiple on-policy learned Options can enable a deep reinforcement learning method to have height reference tracking and online adaptivity in the cases of structural failure of the horizontal tailplane and sign change of pitch damping. In addition, PPOC is able to generalize to a different aircraft by means of transfer learning. Although, the method does not provide sample efficiency benefits over its non-hierarchical counterpart. Unfortunately, PPOC is not suitable for flight control due to its high-frequency control input to the actuators as it leads to reduced durability of the actuators and unwanted vibrations. More research into controlling stochasticity of the stochastic policies and learning of the termination function can lead to more control over the agent's output. Consequently, it should then render the PPOC method suitable for adaptive flight control of novel and traditional aircraft configurations, where ultimately further research is required that includes the effects of actuator dynamics, non-linear aircraft dynamics, and flight tests.

7

Recommendations

The following improvements or directions of further research are recommended.

- The conventional formulation of the Gaussian distribution does not allow for scaling of the variance as it will change the probability distribution. This is troublesome as real-life control applications have physical limits. If the agent can be constrained to these limits from the start, then the agent might learn faster and have a physically correct behavior. A method that can aid the agent's action selection, in this regard, is the Beta distribution [11]. Research into this method might provide for a more capable PPO or PPOC method for flight control.

- Entropy contributes to the stochastic behavior of the agent but also contributes to more optimal policies as it keeps exploring the state space. Though in practice, the entropy bonus is turned off for online applications as it can cause sudden drops in performance. Adaptively varying the entropy can improve adaptivity while retaining online performance, as it controls the level of stochasticity of a stochastic policy. A learned entropy parameter already has proven its success. The Soft Actor-Critic (SAC) [21] is an example of such a method. Although SAC is an off-policy method, it has already proven its success as a control method that can work robustly in an online setting [12]. Research into applying a learned entropy parameter to PPOC can reduce high-frequency control output, improve reference tracking through having more control on the agent's output.

- Sample efficiency can be improved by letting the MLP of the value and policy function share parameters. This allows for sharing a baseline knowledge of the environment. The MLP ends with two heads, one for the value function and the other for the policy function. This should greatly improve sample efficiency but can introduce bias. This method was proven for various non-hierarchical reinforcement learning methods in [26] but has not been applied to an Option-Critic architecture.

- A method that might improve and, at the same, provide for a smoother reference tracking is the use of an initiation set. The Option-Critic method only learns the termination function, intra-option policy, and policy over options and does not use the initiation set by assuming that all Options are available for all states. As a consequence and also observed during the preliminary analysis, the Options will switch quite frequently, which leads to high-frequency control and spiky reference tracking. If the Options could be prolonged, this might reduce the spiky reference tracking as their behavior starts to resemble that of PPO, which proved to have smoother reference tracking.

- Another method that can result in more control over the switching induced by the termination function is the Option-Critic with a deliberation cost [22]. The deliberation cost is added to the termination gradient, where it lets the agent switch less frequently by incurring a cost for switching.

- In this research, the offline training only used a sine signal, a more elaborate training scheme can contribute to a more complex and capable controller. The PPOC agent can be encouraged to display more complex behavior by exposing it to different initial conditions, adaptivity tests, and curriculum learning strategies during offline training.

- For flight control, it is desired to have a method that is able to continually adapt to changing internal or external circumstances, a field that seeks a solution for that is continual learning. A way to create continual learning aspects with the Option-Critic would be to log the tracking error for an extended period of time and track its standard deviation. If the standard deviation goes over a specified limit, then the Option-Critic should activate its full learning potential. If not, it should be dormant. There are many ways to activate the learning potential. One way would be to design a learning rate scheme that is coupled to the standard deviation, or another way would be to fixate one Option when the agent is dormant and enable all Options when the agent needs full learning capability or needs recovery capabilities. This allows the PPOC agent to harness the sample efficiency of a single policy and benefit from the adaptivity in situations when it is needed.
- PPOC shows promise that it can learn similar or better online learning behavior as IDHP. PPOC does not require an incremental model estimation, whereas IDHP does. Though turning PPOC into an incremental model-dependent method will most likely improve the performance with a great factor, as a general rule, methods that use model information during the learning process are often outperforming model-independent methods.
- The use of direct value gradient estimation as opposed to indirectly estimating the value gradient by first estimating the value function and then differentiating it should also greatly improve learning. A method using direct value gradient estimation might require fewer samples as it has more information. This intuition for faster learning and better learning was proven in [45] by identifying the aforementioned intuition as the main difference between HDP and DHP methods. In addition, the IDHP method benefitted from using reward gradients directly as opposed to PPOC, where it mainly received a scalar reward computed from the non-differentiated reward function. Passing the scalar value of the reward gradient to the agent might have provided the agent with some foresight on future value. The use of reward gradient and estimating value gradients goes hand-in-hand.

IV

Appendices

A

Mass-spring-damper model

The mass-spring-damper (MSD) model is taken from [10]. The equations of motion for MSD model with three masses is given in eq. (A.1). The coefficients that are used for this research are provided in table A.1.

$$\begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{-(k_1+k_2)}{m_1} & \frac{k_2}{m_1} & 0 & \frac{-(c_1+c_2)}{m_1} & \frac{c_2}{m_1} & 0 \\ \frac{k_2}{m_2} & \frac{-(k_2+k_3)}{m_2} & \frac{k_3}{m_2} & \frac{c_2}{m_2} & \frac{-(c_2+c_3)}{m_2} & \frac{c_3}{m_2} \\ 0 & \frac{k_3}{m_3} & \frac{-k_3}{m_3} & 0 & \frac{c_3}{m_3} & \frac{-c_3}{m_3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} + \begin{bmatrix} \frac{1}{m_1} & 0 & 0 \\ 0 & \frac{1}{m_2} & 0 \\ 0 & 0 & \frac{1}{m_3} \end{bmatrix} \cdot \begin{bmatrix} F_{u_1} \\ F_{u_2} \\ F_{u_3} \end{bmatrix} \tag{A.1}$$

Table A.1: The coefficients that are used for the complete mass-spring-damper system.

Mass	m[kg]	k[N/m]	c $\left[\mathrm{N}/\left(\mathrm{ms}^{-1}\right)\right]$
1	0.4	4	3
2	0.8	3	1
3	0.3	1	6

B

Flight model

The flight models used for the scientific article in part I are presented here. The equations of motion in eq. (B.1) appendix B and stability & control derivatives in appendix B appendix B are taken from [39].

$$\begin{bmatrix} \dot{\hat{u}} \\ \dot{\alpha} \\ \dot{\theta} \\ \frac{\dot{q}\bar{c}}{V} \end{bmatrix} = \begin{bmatrix} x_u & x_\alpha & x_\theta & 0 \\ z_u & z_\alpha & z_\theta & Z_q \\ 0 & 0 & 0 & \frac{V}{c} \\ m_u & m_\alpha & m_\theta & m_q \end{bmatrix} \begin{bmatrix} \hat{u} \\ \alpha \\ \theta \\ \frac{q\bar{c}}{V} \end{bmatrix} + \begin{bmatrix} x_{\delta_e} & x_{\delta_t} \\ z_{\delta_e} & z_{\delta_t} \\ 0 & 0 \\ m_{\delta_e} & m_{\delta_t} \end{bmatrix} \begin{bmatrix} \delta_e \\ \delta_t \end{bmatrix} \tag{B.1}$$

Table B.1: Linearized equations of motions for symmetric flight

	$x_{...}$	$z_{...}$	$m_{...}$
u	$\frac{V}{\bar{c}}\frac{C_{X_u}}{2\mu_c}$	$\frac{V}{\bar{c}}\frac{C_{Z_u}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$\frac{V}{\bar{c}}\frac{C_{m_u}+C_{Z_u}\frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$
α	$\frac{V}{\bar{c}}\frac{C_{X_\alpha}}{2\mu_c}$	$\frac{V}{\bar{c}}\frac{C_{Z_\alpha}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$\frac{V}{\bar{c}}\frac{C_{m_\alpha}+C_{Z_\alpha}\frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$
θ	$\frac{V}{\bar{c}}\frac{C_{Z_0}}{2\mu_c}$	$-\frac{V}{\bar{c}}\frac{C_{X_0}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$-\frac{V}{\bar{c}}\frac{C_{X_0}\frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$
q	$\frac{V}{\bar{c}}\frac{C_{X_q}}{2\mu_c}$	$\frac{V}{\bar{c}}\frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$\frac{V}{\bar{c}}\frac{C_{m_q}+C_{m_{\dot{\alpha}}}\frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$
δ_e	$\frac{V}{\bar{c}}\frac{C_{X_{\delta_e}}}{2\mu_c}$	$\frac{V}{\bar{c}}\frac{C_{Z_{\delta_e}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$\frac{V}{\bar{c}}\frac{C_{m_{\delta_e}}+C_{Z_{\delta_e}}\frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$
δ_t	$\frac{V}{\bar{c}}\frac{C_{X_{\delta_t}}}{2\mu_c}$	$\frac{V}{\bar{c}}\frac{C_{Z_{\delta_t}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$\frac{V}{\bar{c}}\frac{C_{m_{\delta_t}}+C_{Z_{\delta_t}}\frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$

Table B.2: Symmetric stability and control derivatives for a jet aircraft during cruise

V	=	125.675 m/sec	m	= -	kg	$\bar{c}$	=	2.057 m
S	=	m^2	l_h	= -	m	μ_c	=	89.7
K_Y^2	=	1.4996	x_{cg}	= -	$\bar{c}$			
C_{X_0}	=	0	C_{Z_0}	=	-0.2292			
C_{X_u}	=	-0.0032	C_{Z_u}	=	-0.4592	C_{m_u}	=	0.0236
C_{X_α}	=	0.1692	C_{Z_α}	=	-5.7874	C_{m_α}	=	-0.7486
$C_{X_{\dot{\alpha}}}$	=	0	$C_{Z_{\dot{\alpha}}}$	=	-4.2255	$C_{m_{\dot{\alpha}}}$	=	-1.650
C_{X_q}	=	-0.0450	C_{Z_q}	=	-4.5499	C_{m_q}	=	-7.4647
$C_{X_{\delta_e}}$	=	0	$C_{Z_{\delta_e}}$	=	-0.5798	$C_{m_{\delta_e}}$	=	-1.4440

Table B.3: Symmetric stability and control derivatives for a jet aircraft during approach

V	=	67.36 m/sec	m	=	255830 kg	$\bar{c}$	=	8.321 m
S	=	510.97 m^2	l_h	=	31.09 m	μ_c	=	49.12
K_Y^2	=	2.3345	x_{cg}	=	0.25 $\bar{c}$			
C_{X_0}	=	0	C_{Z_0}	=	-1.760			
C_{X_u}	=	0	C_{Z_u}	=	-3.3	C_{m_u}	=	0.071
C_{X_α}	=	0.630	C_{Z_α}	=	-5.933	C_{m_α}	=	-1.450
$C_{X_{\dot{\alpha}}}$	=	0	$C_{Z_{\dot{\alpha}}}$	=	-3.350	$C_{m_{\dot{\alpha}}}$	=	-1.650
C_{X_q}	=	0	C_{Z_q}	=	-2.825	C_{m_q}	=	-10.70
$C_{X_{\delta_e}}$	=	0	$C_{Z_{\delta_e}}$	=	-0.360	$C_{m_{\delta_e}}$	=	-1.400

C

Additional figures

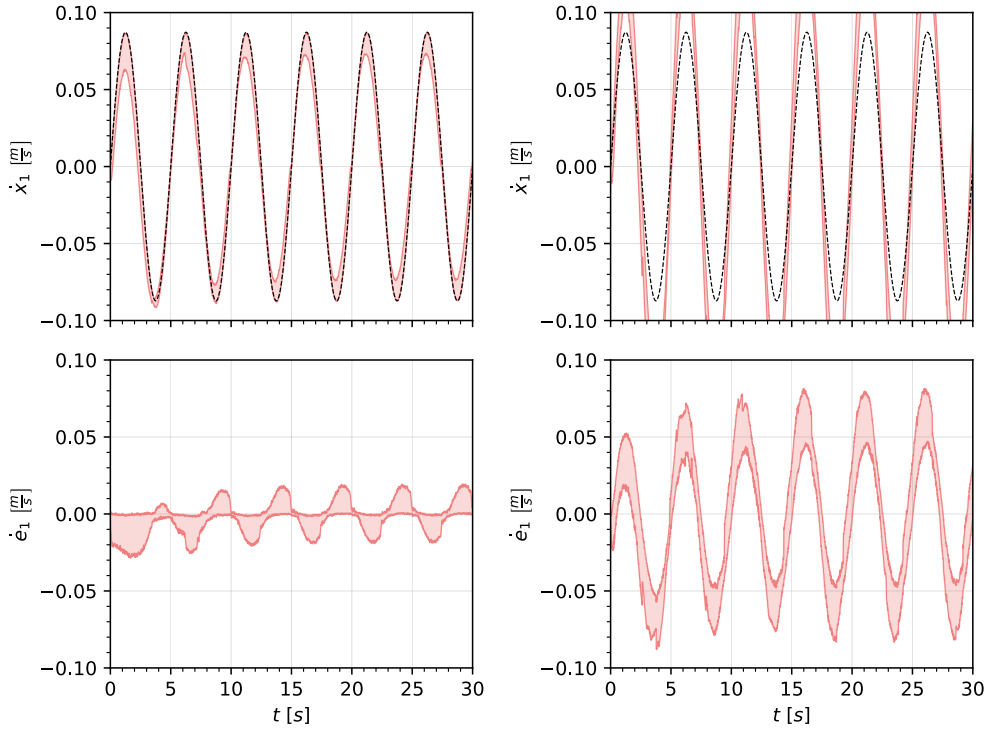

Figure C.1: Results of velocity tracking in experiment 1 with the damping constant adaptivity test for PPO. In the left and right column are the time traces before the test and after the test, respectively. See fig. 5.14 for the corresponding learning curves.

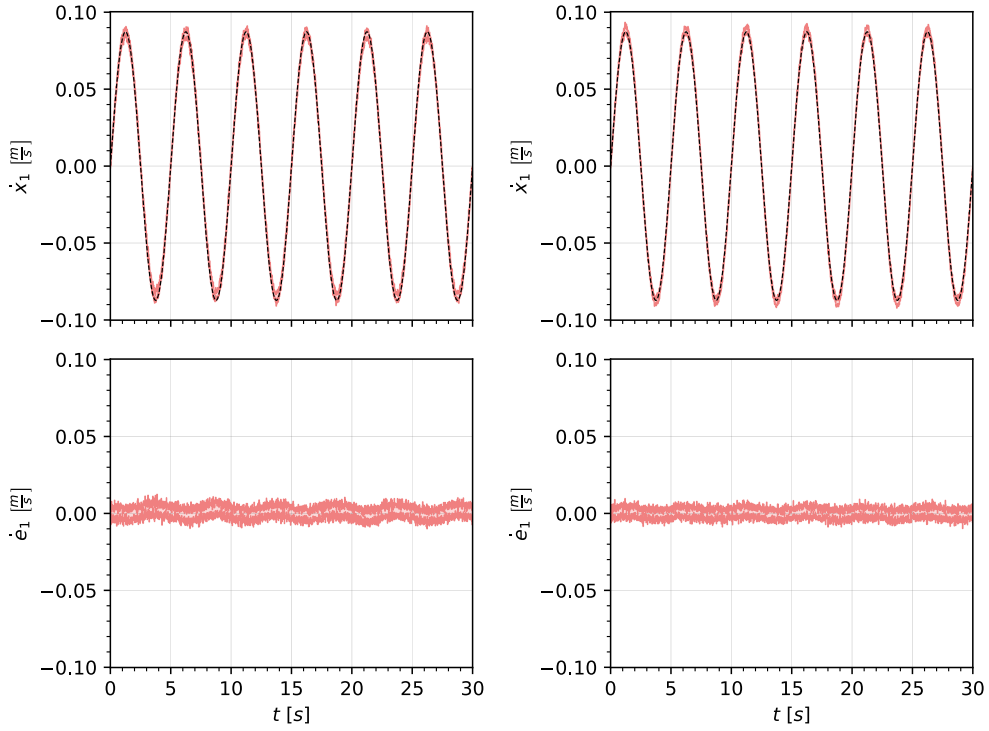

Figure C.2: Results of velocity tracking in experiment 1 with the damping constant adaptivity test for PPOC-4. In the left and right column are the time traces before the test and after the test, respectively. See fig. 5.14 for the corresponding learning curves.

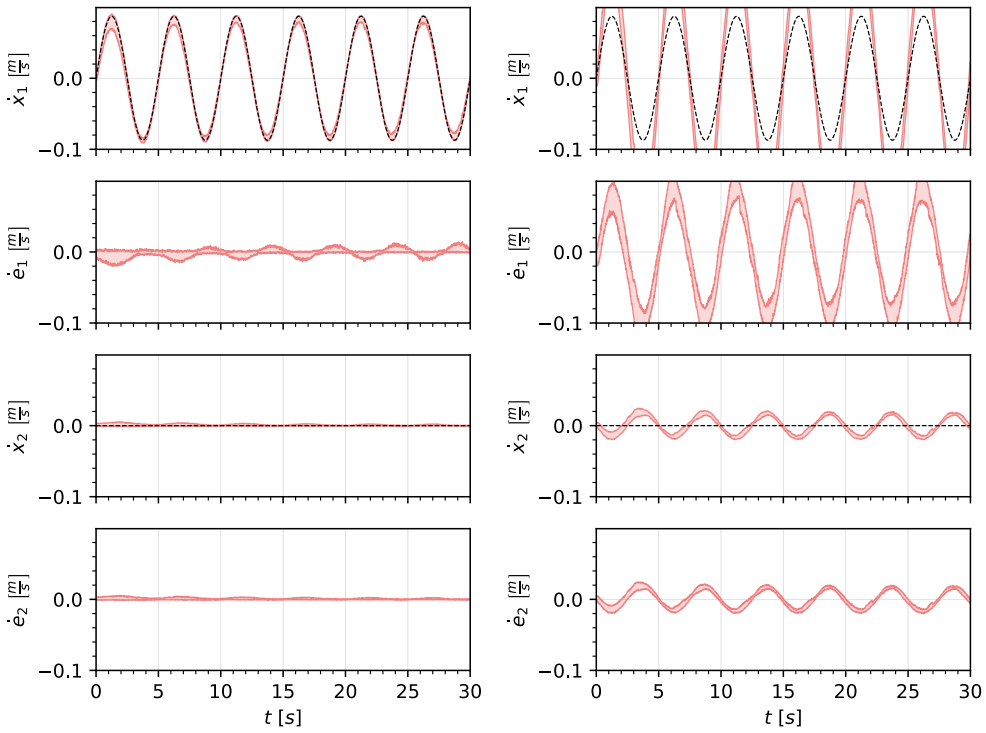

Figure C.3: Results of velocity tracking in experiment 2 with the damping constant adaptivity test for PPO. In the left and right column are the time traces before the test and after the test, respectively. See fig. 5.15 for the corresponding learning curves.

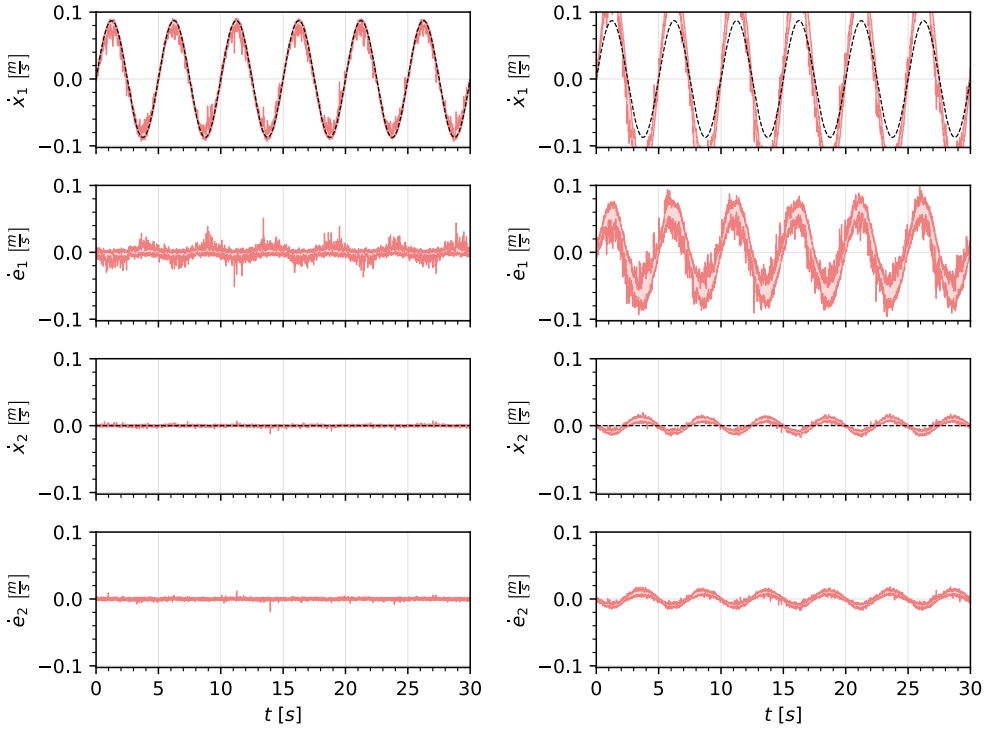

Figure C.4: Results of velocity tracking in experiment 2 with the damping constant adaptivity test for PPOC-2. In the left and right column are the time traces before the test and after the test, respectively. See fig. 5.15 for the corresponding learning curves.

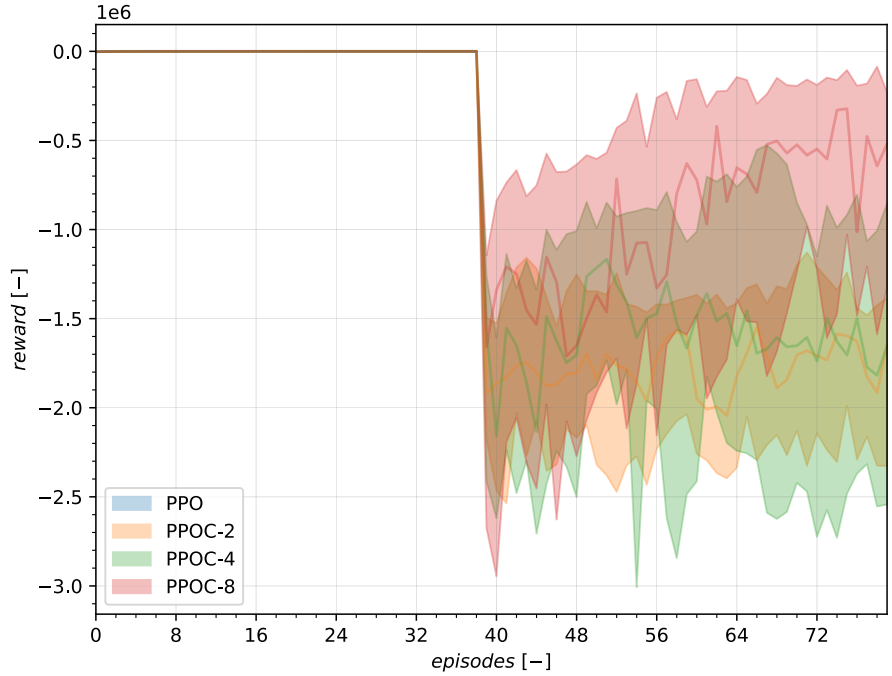

Figure C.5: EXP1, velocity tracking with k-test.

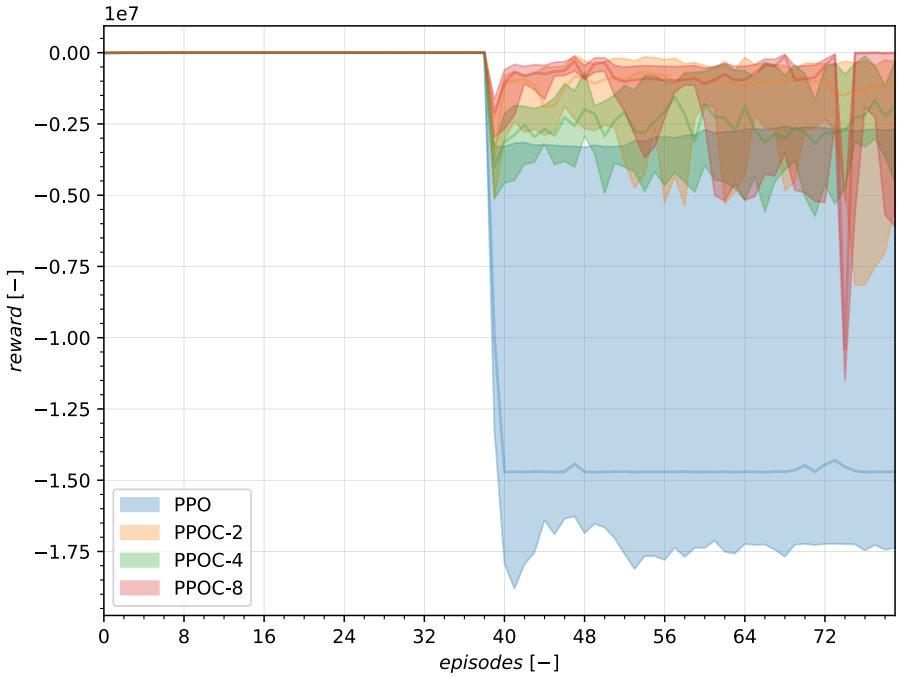

Figure C.6: EXP2, velocity tracking with k-test.

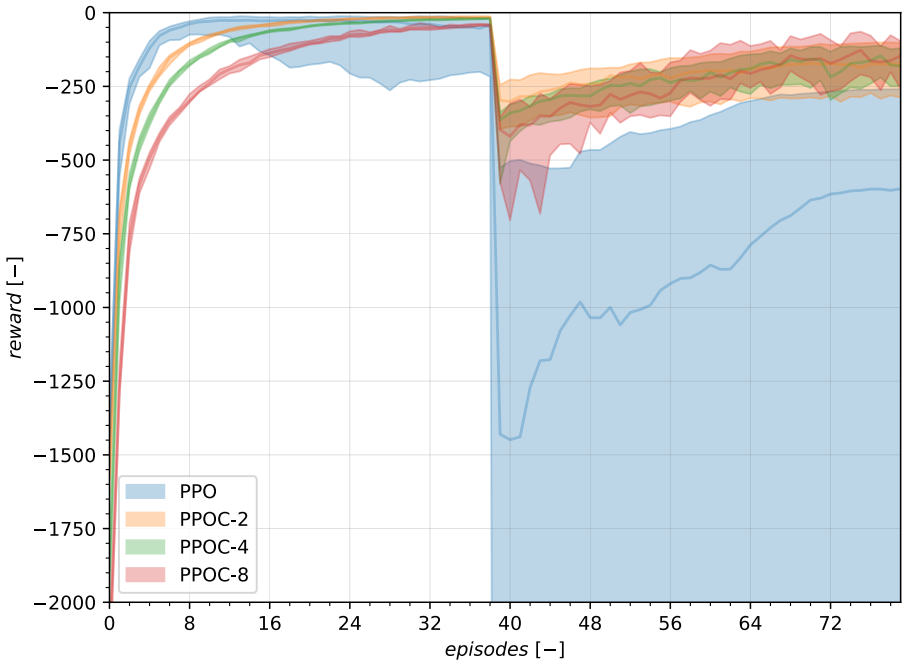

Figure C.7: EXP3, velocity tracking with k-test.

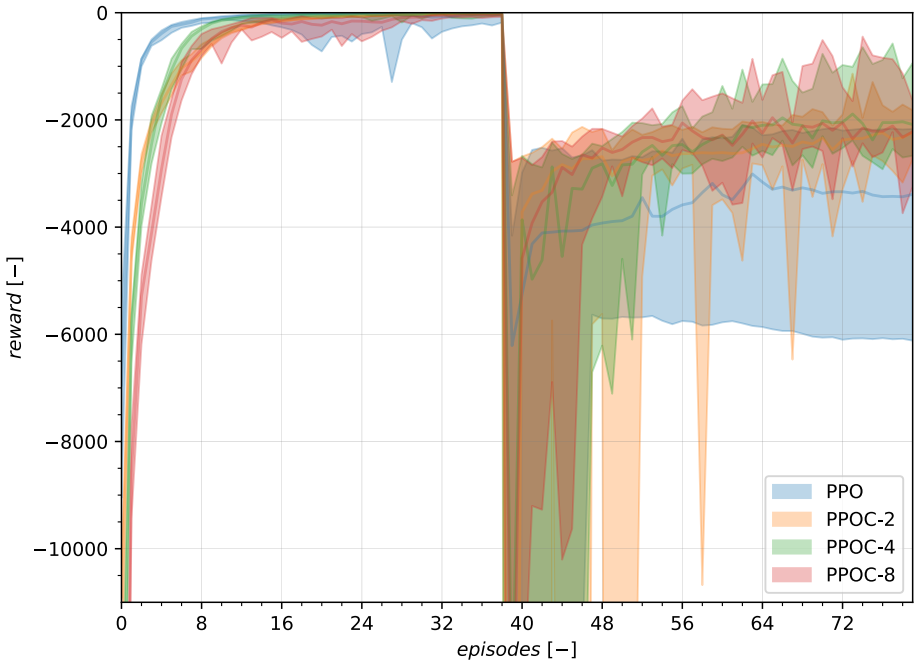

Figure C.8: EXP4, velocity tracking with k-test.

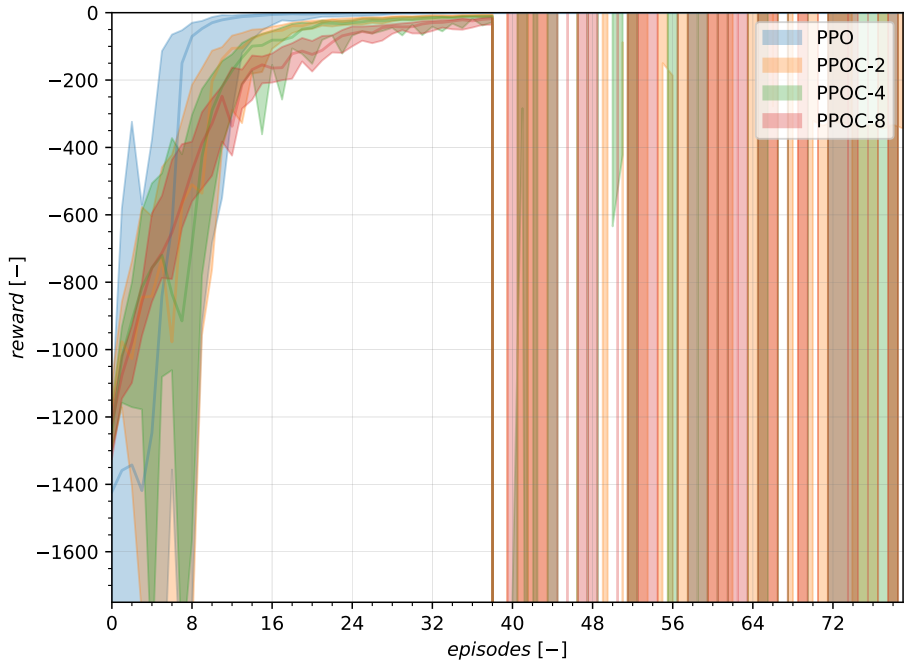

Figure C.9: EXP1, position tracking with c-test.

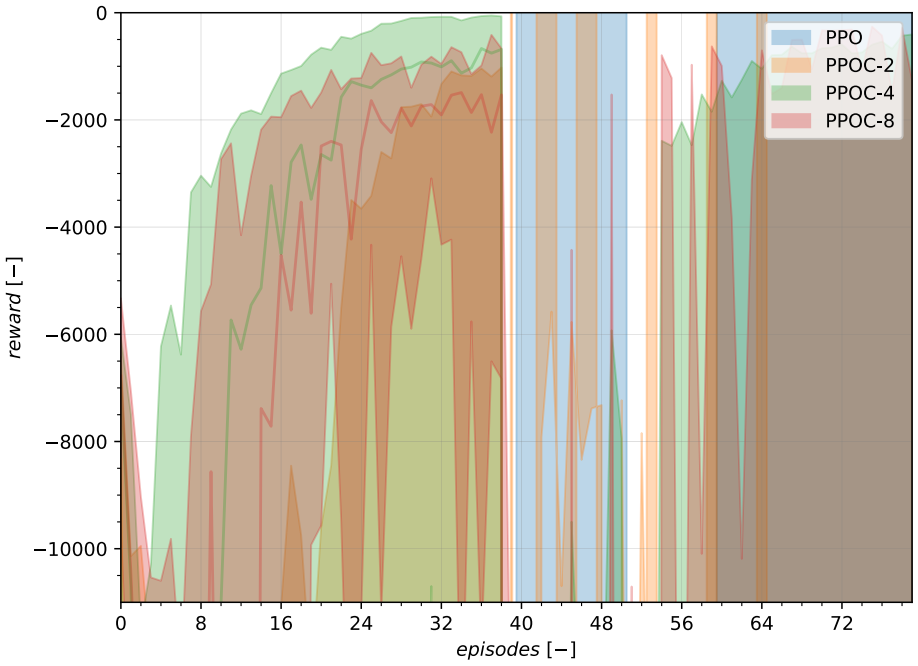

Figure C.10: EXP2, position tracking with c-test.

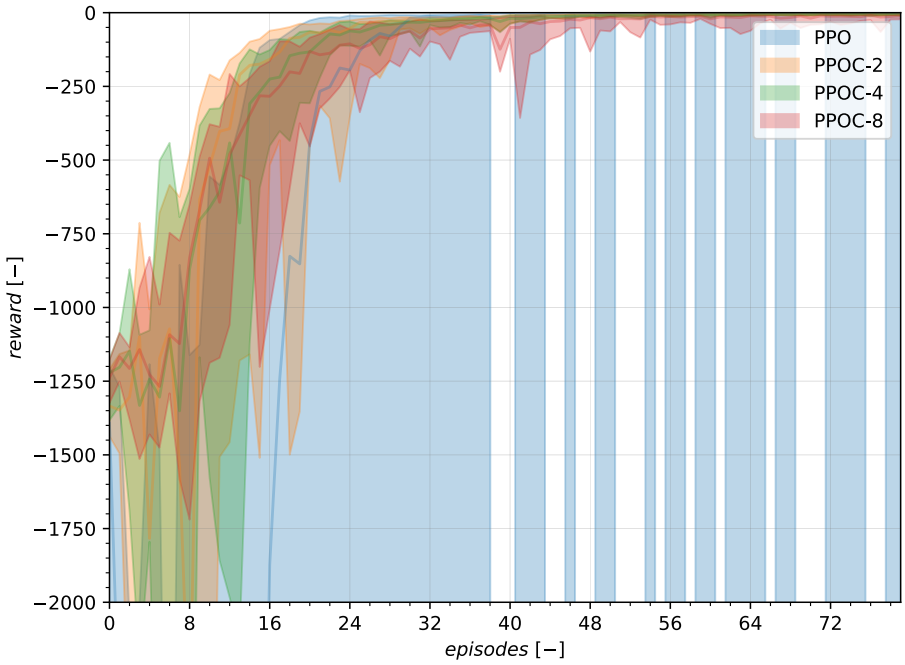

Figure C.11: EXP3, position tracking with c-test.

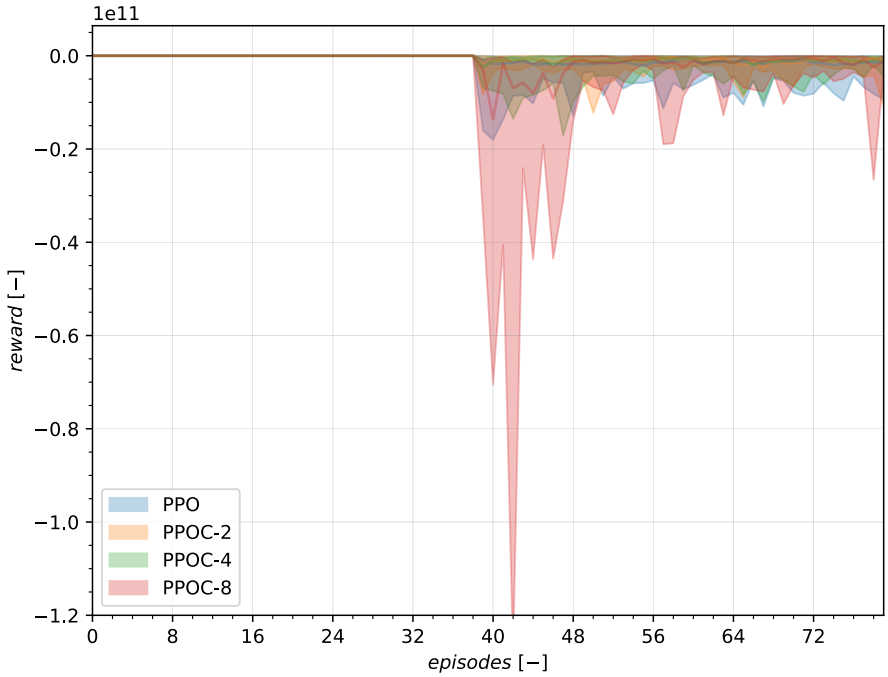

Figure C.12: EXP4, position tracking with c-test.

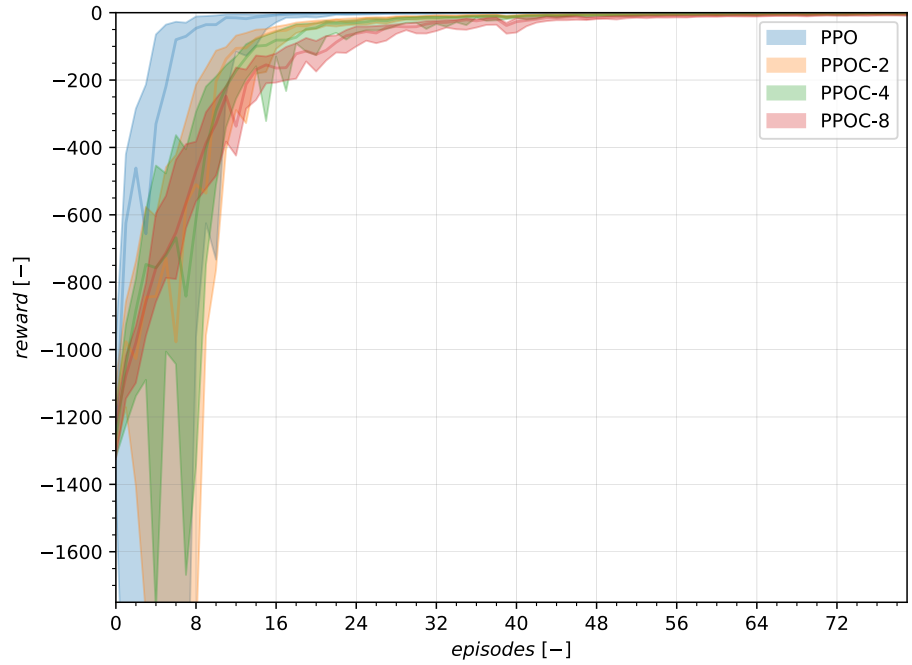

Figure C.13: EXP1, position tracking with k-test.

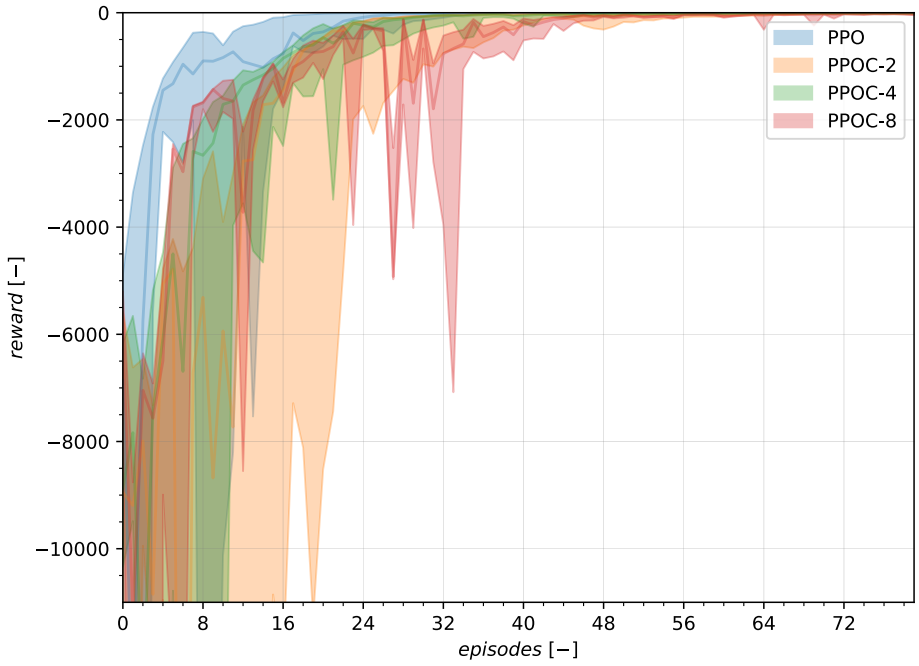

Figure C.14: EXP2, position tracking with k-test.

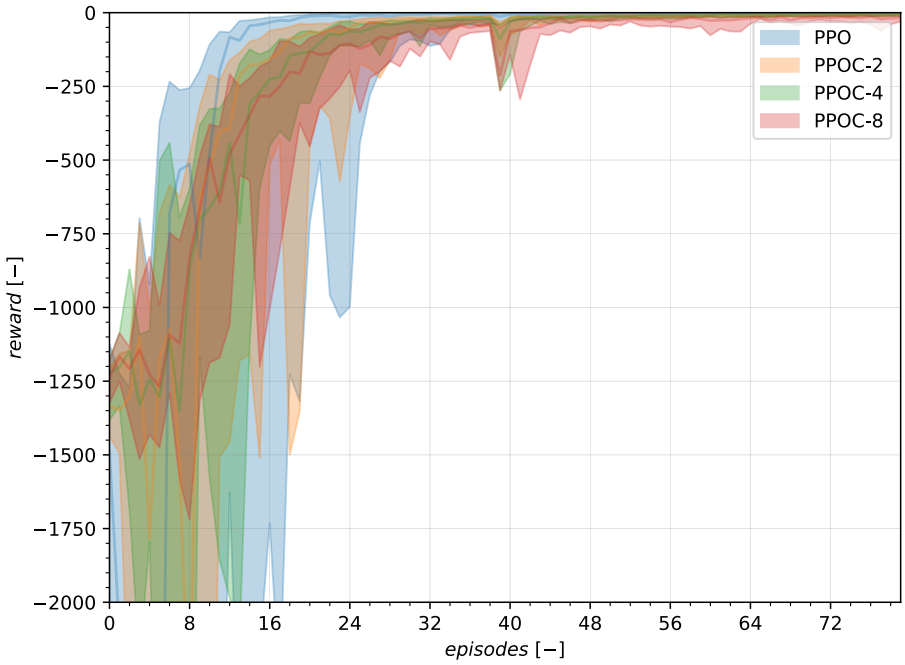

Figure C.15: EXP3, position tracking with k-test.

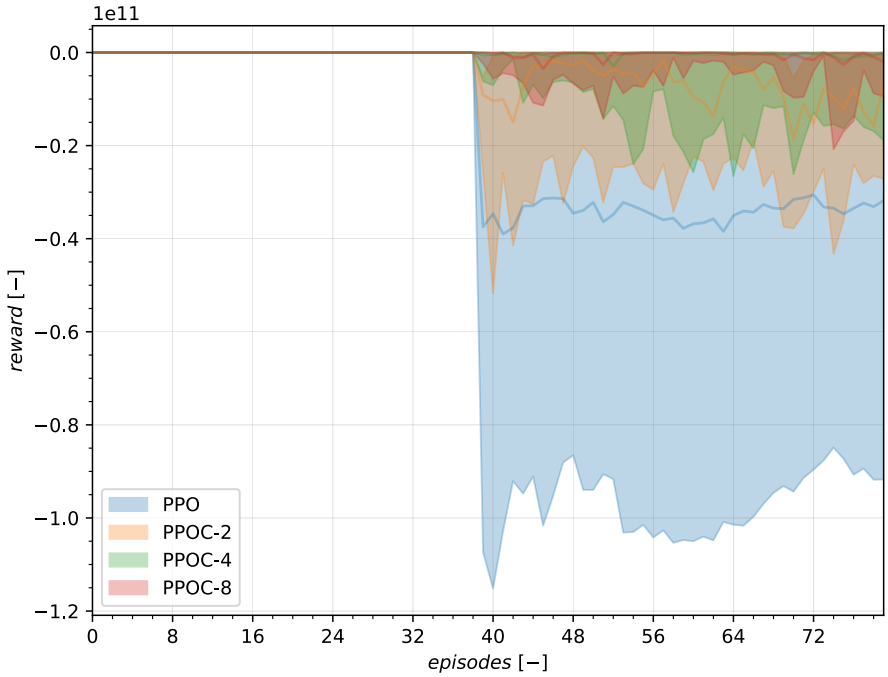

Figure C.16: EXP4, position tracking with k-test.

Bibliography

[1] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:215–239, 2019.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330701. URL https://doi.org/10.1145/3292500.3330701.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5055–5065, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[5] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1–2):41–77, January 2003. ISSN 0924-6703. doi: 10.1023/A:1022140919877. URL https://doi.org/10.1023/A:1022140919877.

[6] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[7] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993.

[8] Clark Borst. AE4316 Aerospace Human-Machine Systems. Technical report, Delft University of Technology, 2018.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv: 1606.01540*, pages 1–4, 2016. URL http://arxiv.org/abs/1606.01540.

[10] De Buysscher. Safe curriculum learning for linear systems with unknown dynamics in primary flight control. Technical report, Delft University of Technology, 2021.

[11] Po Wei Chou, Daniel Maturana, and Sebastian Scherer. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. *34th International Conference on Machine Learning, ICML 2017*, 2:1386–1396, 2017.

[12] Killian Dally. Deep Reinforcement Flight Control Learning for Flight Control. Technical report, Delft University of Technology, 2021.

[13] Peter Dayan and Geoffrey Hinton. Feudal Reinforcement Learning. *Advances in neural information processing systems*, pages 271–278, 1993. ISSN 1049-5258. URL http://www.cs.utoronto.ca/$\sim$hinton/absps/dh93.pdf.

[14] Thomas Degris, Patrick M. Pilarski, and Richard S. Sutton. Model-Free reinforcement learning with continuous action in practice. *Proceedings of the American Control Conference*, pages 2177–2182, 2012. ISSN 07431619. doi: 10.1109/acc.2012.6315022.

[15] Pedro Miguel Dias, Ye Zhou, and Erik Jan van Kampen. Intelligent nonlinear adaptive flight control using incremental approximate dynamic programming. In *AIAA Scitech 2019 Forum*, Reston, Virginia, jan 2019. American Institute of Aeronautics and Astronautics. ISBN 9781624105784. doi: 10.2514/6.2019-2339. URL https://arc.aiaa.org/doi/10.2514/6.2019-2339.

[16] Thomas G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 2000. ISSN 10769757. doi: 10.1613/jair.639.

[17] Thomas G Dietterich. An overview of MAXQ hierarchical reinforcement learning. In *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, volume 1864, pages 26–44, 2000. ISBN 3540678395. doi: 10.1007/3-540-44914-0_2.

[18] R. Enns and Jennie Si. Helicopter trimming and tracking control using direct neural dynamic programming. *IEEE Transactions on Neural Networks*, 14(4):929–939, 2003. doi: 10.1109/TNN.2003.813839.

[19] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning, 2017. ISSN 23318422.

[20] Mohammad Ghavamzadeh and Sridhar Mahadevan. Hierarchical Policy Gradient Algorithms. *Proceedings, Twentieth International Conference on Machine Learning*, 1:226–233, 2003.

[21] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *35th International Conference on Machine Learning, ICML 2018*, 5:2976–2989, 2018.

[22] Jean Harb, Pierre Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option: Learning options with a deliberation cost. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3165–3172, 2018.

[23] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3207–3214, 2018.

[24] S. Heyer. Reinforcement learning for flight control. Technical report, Delft University of Technology, 2019. URL http://resolver.tudelft.nl/uuid:dc63cae7-4289-47c7-889e-253f7abd7c72.

[25] S. Heyer, D. Kroezen, and E. van Kampen. Online adaptive incremental reinforcement learning flight control for a cs-25 class aircraft. In *AIAA Scitech 2020 Forum*, volume AIAA 2020-1844, Reston, Virginia, jan 2020. American Institute of Aeronautics and Astronautics. ISBN 9781624105951. doi: 10.2514/6.2020-1844. URL http://resolver.tudelft.nl/uuid:38547b1d-0535-4b30-a348-67ac40c7ddcchttps://arc.aiaa.org/doi/10.2514/6.2020-1844.

[26] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[27] Sham Kakade. A natural policy gradient. *Advances in Neural Information Processing Systems*, 2002. ISSN 10495258.

[28] Bahare Kiumarsi, Kyriakos G. Vamvoudakis, Hamidreza Modares, and Frank L. Lewis. Optimal and Autonomous Control Using Reinforcement Learning: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6):2042–2062, 2018. ISSN 21622388. doi: 10.1109/TNNLS.2017.2773458.

[29] Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. Learnings options end-to-end for continuous action tasks. *arXiv preprint arXiv: 1712.00004*, 2017. URL http://arxiv.org/abs/1712.00004.

[30] Jun Hyeon Lee. Online reinforcement learning for fixed-wing aircraft longitudinal control. Technical report, Delft University of Technology, Delft, 2019. URL http://resolver.tudelft.nl/uuid:c1201f27-964c-4257-ad65-89224bef94a1.

[31] Doug J. Leith and W. E. Leithead. Survey of gain-scheduling analysis and design. *International Journal of Control*, 73(11):1001–1025, 2000. ISSN 00207179. doi: 10.1080/002071700411304.

[32] Tingguang Li, Jin Pan, Delong Zhu, and Max Q.H. Meng. Learning to Interrupt: A Hierarchical Deep Reinforcement Learning Framework for Efficient Exploration. *2018 IEEE International Conference on Robotics and Biomimetics, ROBIO 2018*, pages 648–653, 2018. doi: 10.1109/ROBIO.2018.8665177.

[33] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[34] Thomas Lombaerts, John Kaneshige, Stefan Schuet, Gordon Hardy, Bimal Aponso, and Kimberlee Shish. Dynamic inversion based full envelope flight control for an eVTOL vehicle using a unified framework. *AIAA Scitech 2020 Forum*, 1 PartF(January):1–30, 2020. doi: 10.2514/6.2020-1619.

[35] Peng Lu, Erik Jan van Kampen, Cornelis de Visser, and Qiping Chu. Aircraft fault-tolerant trajectory control using Incremental Nonlinear Dynamic Inversion. *Control Engineering Practice*, 57:126–141, 2016. ISSN 09670661. doi: 10.1016/j.conengprac.2016.09.010. URL http://dx.doi.org/10.1016/j.conengprac.2016.09.010.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv: 1312.5602*, dec 2013. URL http://arxiv.org/abs/1312.5602.

[37] Volodymyr Mnih, Adria Puigdomenech Badia, Lehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016*, 4:2850–2869, 2016.

[38] Gabriel Moser. In Memoriam, Arthur Samuel: Pioneer in Machine Learning. *AI Magazine (AAAI)*, 11(3):10–11, 1990. ISSN 0018-8646.

[39] J. A. Mulder, W. H. J. J. Van Staveren, J. C. Van Der Vaart, E. De Weerdt, C. C. De Visser, A. C. In 't Veld, and E. Mooij. Lecture notes ae3202 flight dynamics. Technical report, Delft University of Technology, 2013.

[40] Ofir Nachum, Honglak Lee, Shixiang Gu, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 2018-Decem(Nips):3303–3313, 2018. ISSN 10495258.

[41] Zhen Ni, Haibo He, Xiangnan Zhong, and Danil V. Prokhorov. Model-Free Dual Heuristic Dynamic Programming. *IEEE Transactions on Neural Networks and Learning Systems*, 26(8):1834–1839, 2015. ISSN 21622388. doi: 10.1109/TNNLS.2015.2424971.

[42] Takayuki Osa, Voot Tangkaratt, and Masashi Sugiyama. Hierarchical reinforcement learning via advantage-weighted information maximization. In *International Conference on Learning Representations*, 2018.

[43] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, pages 1043–1049, 1998. ISBN 0262100762.

[44] Warren B Powell. *Approximate Dynamic Programming*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, USA, aug 2011. ISBN 9781118029176. doi: 10.1002/9781118029176. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84949764394&doi=10.1002%2F9781118029176&partnerID=40&md5=980a63d0f8affd8ab88a62de9fcce024http://doi.wiley.com/10.1002/9781118029176.

[45] D.V. Prokhorov and D.C. Wunsch. Adaptive critic designs. *IEEE Transactions on Neural Networks*, 8(5):997–1007, sep 1997. ISSN 1045-9227. doi: 10.1109/72.623201. URL https://ieeexplore.ieee.org/document/623201/.

[46] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards Generalization and Simplicity in Continuous Control. In *31st Conference on Neural Information Processing Systems*, 2017.

[47] Matthew Riemer, Miao Liu, and Gerald Tesauro. Learning abstract options. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10445–10455, Red Hook, NY, USA, 2018. Curran Associates Inc.

[48] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. *32nd International Conference on Machine Learning, ICML 2015*, 3:1889–1897, 2015.

[49] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pages 1–14, 2016.

[50] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv: 1707.06347*, pages 1–12, 2017. URL `http://arxiv.org/abs/1707.06347`.

[51] Haibo Shi, Yaoru Sun, Guangyuan Li, Fang Wang, Daming Wang, and Jie Li. Hierarchical intermittent motor control with deterministic policy gradient. *IEEE Access*, 7:41799–41810, 2019. ISSN 21693536. doi: 10.1109/ACCESS.2019.2904910.

[52] S. Sieberling, Q. P. Chu, and J. A. Mulder. Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction. *Journal of Guidance, Control, and Dynamics*, 33(6):1732–1742, 2010. ISSN 15333884. doi: 10.2514/1.49978.

[53] Christopher Silva, Wayne Johnson, Kevin R. Antcliff, and Michael D. Patterson. VTOL urban air mobility concept vehicles for technology development. *2018 Aviation Technology, Integration, and Operations Conference*, pages 1–16, 2018. doi: 10.2514/6.2018-3847.

[54] David Silver. Lecture 1 : Introduction to Reinforcement Learning Outline. Technical report, University College London, 2015.

[55] David Silver. Lecture 7 : Policy gradient. Technical report, University College London, 2015.

[56] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. *31st International Conference on Machine Learning, ICML 2014*, 1: 605–619, 2014.

[57] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 2017. ISSN 14764687. doi: 10.1038/nature24270.

[58] P. Simplício, M. D. Pavel, E. van Kampen, and Q. P. Chu. An acceleration measurements-based approach for helicopter nonlinear flight control using incremental nonlinear dynamic inversion. *Control Engineering Practice*, 21(8):1065–1077, 2013. ISSN 09670661. doi: 10.1016/j.conengprac.2013.03.009. URL `http://dx.doi.org/10.1016/j.conengprac.2013.03.009`.

[59] R S Sutton, D Precup, and S Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *ARTIFICIAL INTELLIGENCE*, 112(1-2):181–211, aug 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00052-1.

[60] Richard S. Sutton. Introduction: The challenge of reinforcement learning. *Machine Learning*, 8(3-4): 225–227, 1992. ISSN 08856125. doi: 10.1007/BF00992695.

[61] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[62] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Funciton Approximation. *Advances in Neural Information Processing Systems*, 70(9), 2000. ISSN 00334081.

[63] Sana Syed, Z. H. Khan, M. Salman, Usman Ali, and Arsalan Aziz. Adaptive flight control of an aircraft with actuator faults. *2014 International Conference on Robotics and Emerging Allied Technologies in Engineering, iCREATE 2014 - Proceedings*, pages 249–254, 2014. doi: 10.1109/iCREATE.2014.6828374.

[64] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. *IEEE International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. ISSN 21530858. doi: 10.1109/IROS.2012.6386109.

[65] E. Van Kampen, Q. P. Chu, and J. A. Mulder. Continuous adaptive critic flight control aided with approximated plant dynamics. *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2006*, 5(August):2989–3016, 2006. doi: 10.2514/6.2006-6429.

[66] Erik-Jan Van Kampen. AE4301 Automatic Flight Control System Design. Technical report, Delft University of Technology, 2018.

[67] Erik-Jan Van Kampen. AE4350 Bio-inspired Intelligence and Learning. Technical report, Delft University of Technology, Delft, 2019.

[68] Ganesh K. Venayagamoorthy, Ronald G. Harley, and Donald C. Wunsch. Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator. *IEEE Transactions on Neural Networks*, 13(3):764–773, 2002. ISSN 10459227. doi: 10.1109/TNN.2002.1000146.

[69] WallPaperAccess. Paper plane wallpapers, 2021. URL `wallpaperaccess.com/paper-plane#1981484`.

[70] Christopher J C H Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

[71] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, may 1992. ISSN 0885-6125. doi: 10.1007/BF00992698. URL `http://link.springer.com/10.1007/BF00992698`.

[72] R. J. Williams. Simple Stiatistical Gradient-Following Algorithms For Connectionist Reinforcement Learning. *Machine Learning*, 8(3-4):229–256, 1992. ISSN 0885-6125. doi: 10.1007/BF00992696.

[73] Yuhuai Wu, Elman Mansimo, Shun Liao, Alec Radford, and John Schulman. OpenAI Baselines: ACKTR & A2C, 2017. URL `https://openai.com/blog/baselines-acktr-a2c/`.

[74] Zhixiong Xu, Lei Cao, and Xiliang Chen. Learning to Learn: Hierarchical Meta-Critic Networks. *IEEE Access*, 7:57069–57077, 2019. ISSN 21693536. doi: 10.1109/ACCESS.2019.2914469.

[75] Zhaoyang Yang, Kathryn Merrick, Hussein Abbass, and Lianwen Jin. Multi-task deep reinforcement learning for continuous action control. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3301–3307, 2017. doi: 10.24963/ijcai.2017/461. URL `https://doi.org/10.24963/ijcai.2017/461`.

[76] Zhaoyang Yang, Kathryn Merrick, Senior Member, Lianwen Jin, Hussein A Abbass, and Senior Member. Hierarchical Deep Reinforcement Learning for Continuous Action Control. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5174–5184, 2018.

[77] Shangtong Zhang and Shimon Whiteson. Dac: The double actor-critic architecture for learning options. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/4f284803bd0966cc24fa8683a34afc6e-Paper.pdf`.

[78] Ye Zhou, Erik-Jan Van Kampen, and Q Chu. Incremental Model Based Heuristic Dynamic Programming for Nonlinear Adaptive Flight Control. *Proceedings of the international micro air vehicles conference and competition (IMAV) 2016*, 2016.

[79] Ye Zhou, Erik-Jan Van Kampen, and Q Ping Chu. An Incremental Approximate Dynamic Programming Flight Controller Based on Output Feedback. In *AIAA Guidance, Navigation, and Control Conference*, 2016. ISBN 9781624103896. doi: 10.2514/6.2016-0360.

[80] Ye Zhou, Erik Jan Van Kampen, and Qi Ping Chu. Incremental approximate dynamic programming for nonlinear adaptive tracking control with partial observability. *Journal of Guidance, Control, and Dynamics*, 41(12):2554–2567, 2018. ISSN 15333884. doi: 10.2514/1.G003472.