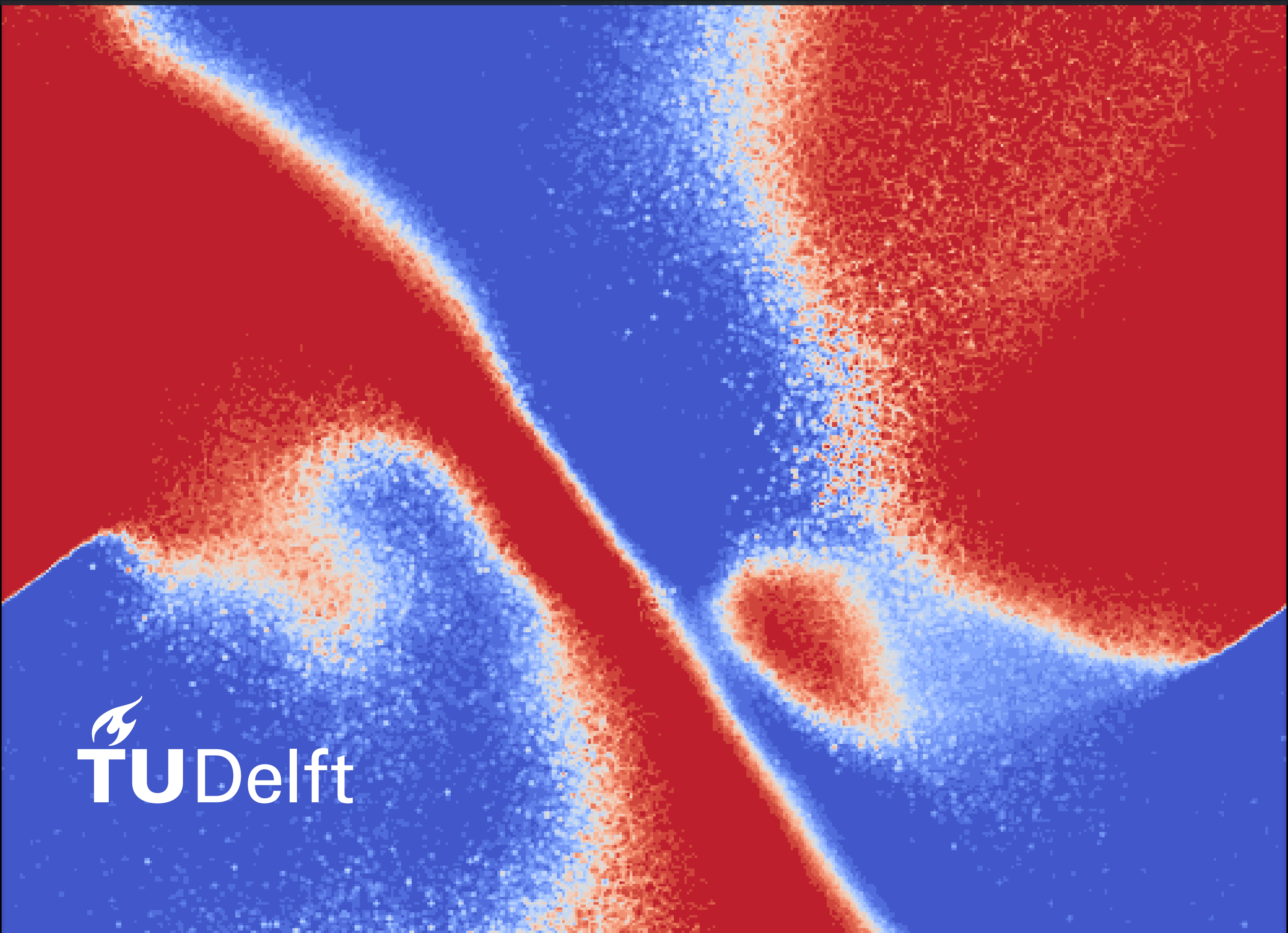
Towards Sample-Efficient Offline Reinforcement Learning in Flight Control

A Study on Sample-Efficient Model-Free Algorithms for Flight Control Tasks

Gabriel Alayón Blanco



TU Delft

Towards Sample-Efficient Offline Reinforcement Learning in Flight Control

A Study on Sample-Efficient Model-Free Algorithms for Flight Control Tasks

Thesis report

by

Gabriel Alayón Blanco

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on July 9, 2025 at 13:30

Thesis committee:	
Chair:	Dr.Ir. C.C de Visser
Supervisors:	Dr.Ir. Erik-Jan van Kampen
	Ir. Isabelle El-Hajj
External examiner:	Dr. O.A. Sharpanskykh
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	June, 2024 - July, 2025
Student number:	4447662

Faculty of Aerospace Engineering · Delft University of Technology

Preface

This master's thesis is the culmination of my tertiary education. Throughout my academic journey, I have learned a lot and not just limited to what was in the curriculum. To draw a parallel with reinforcement learning: I came to TU Delft's Aerospace Engineering as a young man untrained in many ways and through many trials will leave this institution a confident fully trained engineer ready to play their part in helping solve some of the world's problems. I would like to thank my supervisors Erik-Jan and Isabelle for their guidance, expertise and support throughout my thesis. As usual, a special thanks to my family and friends for their continued support and motivation.

For readers interested in the implementation details, the GitHub repository containing the code used to generate the results presented in the scientific article is available at `https://github.com/gabCodes/RL_FlightControl`.

Gabriel Alayón Blanco
Delft, 2025

Contents

Acronyms

AI	Artificial Intelligence.
CAPS	Conditioning for Action Policy Smoothness.
CVaR	Conditional Value at Risk.
DDPG	Deep Proximal Policy Gradient.
DP	Dynamid Programming.
DQL	Double Q-Learning.
DQN	Deep Q Network.
GP	Gaussian Process.
IDHP	Incremental Dual Heuristic Programming.
MBPO	Model Based Policy Optimisation.
MBRL	Model Based Reinforcement Learning.
MC	Monte-Carlo.
MDP	Markov Decision Process.
MFRL	Model Free Reinforcement Learning.
ML	Machine Learning.
PPO	Proximal Policy Optimisation.
REDQ	Randomized Ensenmble Double Q-Learning.
RL	Reinforcement Learning.
RQ	Research Question.
SAC	Soft Actor Critic.
SARSA	State Action Reward State Action.
SB	Stable Baselines.
SVM	Support Vector Machines.
TD	Temporal Difference.
TD3	Twin Delayed Deterministic Policy Gradient.
TPE	Tree-structured Parzen Estimator.
UAV	Unmanned Aerial Vehicle.
UCB	Upper Confidence Bound.
UTD	Update To Data.

Nomenclature

Reinforcement learning sets

$\mathbb{N}$	Natural numbers
$\mathbb{R}$	Real numbers
$A(s)$	Set of possible actions at state s
R	Set of rewards
S	Set of states

Reinforcement learning symbols

Δ	Policy evaluation threshold
ϵ	Prob. of random action / noise
γ	Discount factor
κ/H	temperature coefficient / target entropy
λ	Critic network parameters
$\mathbb{E}$	Expected value
Ω	Actor network parameters
π	Policy
ρ	Soft update coefficient
τ	Step size / Learning rate
a	Action
B	Batch size / mini-batch
D	Replay buffer
f	State transition function
G	Return
J	Objective function
L	Loss function
M	Subset size of critic selection
N	Number of critics
p	MDP dynamics function
q/Q	Action-value function
r	Reward function
r_3	Extended reward function
s	State
T	Final timestep
t	Timestep
v/V	Value function
y	Target value

Aircraft states and actuators

α	Angle of attack [rad or deg]
β	Angle of sideslip [rad or deg]
δ_a	Aileron deflection [rad or deg]
δ_e	Elevator deflection [rad or deg]
δ_r	Rudder deflection [rad or deg]
ϕ	Roll angle [rad or deg]
ψ	Yaw angle [rad or deg]
θ	Pitch angle [rad or deg]
h_e	Geometric altitude [m]
p	Roll rate [rad/s or deg/s]
q	Pitch rate [rad/s or deg/s]
r	Yaw rate [rad/s or deg/s]
T_N	Engine thrust [N]
V_{TAS}	True airspeed [m/s]
x_e	Horizontal position along Earth [m]
y_e	Vertical position along Earth [m]

Preliminary analysis states

β_p	Hinged/Inverted pendulum offset angle [rad]
ω	Hinged/Inverted pendulum angular velocity [rad/s]
F_H	Inverted pendulum applied horizontal force [N]
F_T	Hinged pendulum applied torque [N·m]
V_{cart}	Inverted pendulum horizontal velocity [m/s]
x_p	Hinged pendulum horizontal position [m]
x_{cart}	Inverted pendulum horizontal position [m]
y_p	Hinged pendulum vertical position [m]

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The development of aerial vehicle technology has transformed the way in which humans travel, transport goods and defend themselves. Passenger air transport, which suffered a plunge due to the COVID-19 crisis, has recovered the crisis and reached an all time high as of December 2024 with a forecasted growth of 6.2% in 2025 [1]. Historically, pilots played a crucial role in the operating of manned aerial vehicles across all levels of control. Today, pilots reap the rewards of technological innovations by delegating some of, if not all, control tasks to a flight control system. These flight augmentation and automation systems conventionally operate by using linearised aircraft models to decide on actuator control gains to augment or automate many control tasks. However, due to the linearisation, the control gains calculated are only valid near fixed operation points and so most augmentation/automation systems deploy some form of gain scheduling to obtain the appropriate gain across the vehicle's flight envelope.

Conventional flight control techniques that use linearised models to calculate control gains at fixed operating points present two main issues. Firstly, there is a reliance on having a high fidelity model of the aerial vehicle in order to calculate the necessary control gains which needs to be derived, verified and validated thoroughly due to the expensive and rigorous nature of aviation. This is of particular trouble for experimental and innovative vehicles whose dynamics and designs might differ enough from the other vehicles where there's no easy reference model to begin from. Secondly, control systems that use gain scheduling are confined by the regions which have been scheduled, meaning that if for whatever reason the vehicle finds itself outside this region the augmentation/automation control systems will not be appropriately calibrated. Not only that, but as technology advances and aerial vehicles become more capable their flight envelope is likely to broaden and require more calculations once again.

The need for a high fidelity model and a schedule for control gains across multiple operating points can be solved with reinforcement learning (RL). RL refers to a class of machine learning techniques where an agent acts and explores their environments in clever ways to maximise the notion of cumulative reward. For flight control systems, this notion of reward can refer to how well a certain reference is tracked and the actions of the agent would correspond to the commands given to the actuators which has shown to be achievable via RL [2]. Higher level control tasks like autonomous navigation have also been posed as RL problems successfully [3]. Though some RL methods can make use of models to better the agent's decision making, they are not strictly needed and have been shown to work in a model free fashion [4]. These tend to be simpler to use due to their black box approach as no model identification is necessary but come at the expense of solution opaqueness as inferring knowledge from black box solutions is difficult. Finally, if the agent's learned policy generalises sufficiently, gain scheduling would be unnecessary. This is because the policy could operate across the entire flight envelope.

Naturally, RL methods have their disadvantages. Given their exploratory nature they can be data hungry and unsafe. The data hunger problem is so prevalent across RL that there's a term used to describe the performance of different algorithms in terms of their data needs; this term is the sample efficiency and it is a measure of how many interactions agent-environment interactions are necessary for acceptable performance of the algorithm. Much research has gone into how to improve the sample efficiency of RL as it can lower the training data cost of obtaining good solutions, of particular interest is the REDQ method which has shown impressive model free sample efficiency performance in some benchmarks that beats model based approaches [5]. This thesis will look to help RL find more use in civil aviation flight control tracking applications by making use of more sample efficient developments that have not been tested in the field.

1.2 Research objective & research questions

Now that the research gap and motivation have been established, the direction of the thesis can be locked by formulating the research questions and objectives. The objective of the thesis is encapsulated as follows:

Research Objective

To improve the sample efficiency of model free flight control reinforcement learning approaches.

As the research objective states, knowledge must be acquired in order to propose a methodology that tackles the problem of sample efficiency in RL flight control applications. With the aim of acquiring this knowledge and achieving the research objective, the following three research questions (RQ) and accompanying sub-questions are proposed as areas to be covered in this thesis:

Research Questions

Q1 How can reinforcement learning with a focus on sample efficiency be used for flight control applications?

1. How can RL be used to solve control problems?
2. What are some relevant performance metrics to keep in mind when developing RL for flight control purposes?
3. What is a suitable methodology to tackle sample efficiency improvement?

. . . .

Q2 How can a preliminary analysis investigating sample efficiency be implemented on benchmark environments?

1. What learning environments will be used to test the methodology?
2. What baseline algorithms will be used in the preliminary analysis for comparison?
3. How will the sample efficiency of the preliminary analysis be computed?

. . . .

Q3 How does the proposed methodology perform on the Cessna Citation PH-LAB Model?

1. How can the RL algorithm be interfaced with the PH-LAB model?
2. What algorithms will be used as comparisons in the methodology?
3. How will the proposed methodology improve sample efficiency?
4. What are the safety implications for the proposed methodology?

1.3 Thesis structure

The structure of this thesis is broken into three parts. Part I contains a standalone scientific article detailed the experiment conducted. Part II contains the literature study and preliminary analysis. Part III contains additional results not discussed in the paper. Part IV contains the conclusion and recommendations. The breakdown of the chapters is as follows: Chapter 1 gives the motivation for the research and outlines the research direction via the research objective and questions. Chapter 2 will give the necessary background to understand RL, go over some of the common approaches, discuss the different classifications used in RL taxonomy and introduce some benchmarks. Chapter 3 narrows in on the information necessary to tackle the research objective, extending the general knowledge and tailoring it for the flight control applications. Chapter 4 will give a preliminary analysis of the REDQ method, this was done on the "Pendulum-v1" and "InvertedPendulum-v4" environments to get a sense of the effect of the REDQ algorithm and familiarise with the author with the procedures that will be necessary during the thesis exercise. Chapter 5 summarises the literature study by giving an overview of the literature study and some recommendations on how to connect the knowledge from the study to the thesis assignment. Chapter 6 discusses the hyperparameter tuning procedure done for the flight control attitude tracking agents to ensure both policy smoothness and good tracking performance. Chapter 7 presents some visualisations which were omitted on the research article to prevent clutter, it includes additional information regarding the fault scenarios tested during the experiment. Chapter 8 discusses how the verification and validation of the research experiment was addressed. Chapter 9 will conclude by building on all the knowledge gathered from the thesis in order to answer the research questions. Chapter 10 contains the recommendations suggested by the author. The planning proposed for the literature study and thesis can be seen in Appendix A where the study is broken down into work packages and scheduled as seen in the Gantt chart.

Part I

Scientific Article

Enhancing Sample Efficiency for Offline Reinforcement Learning in Flight Control Applications

G. Alayón Blanco [*]
Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands

Sample efficiency is a critical metric in intelligent control systems as it directly influences the feasibility and effectiveness of learning-based approaches. This paper presents the study of how Randomized Ensemble Double Q-Learning (REDQ), a sample-efficient model free algorithm, can be used in flight control applications. Three controllers were developed for: pitch, roll and combined biaxial attitude tracking tasks and tested on a high fidelity Cessna Citation 550 model. For each control task, three agents were trained offline: two using REDQ enhanced Soft Actor Critic (SAC) architectures and one using a standard SAC architecture for comparison. REDQ agents showed statistically significant improvements in sample efficiency during initial learning. Average accurate tracking convergence (error < $1°$) occurred within 5,500 training steps for pitch, 6,400 for roll and 11,500 for biaxial control. The gains in sample efficiency were shown to have drawbacks in learning stability and robustness when deviated too far from nominal conditions.

Nomenclature

Mathematic notation

$\mathbb{R}$	Real numbers	–
S	Set of states	–
$A(s)$	Possible actions at state $s \in S$	–
f	State transition function	–
r	Reward function	–
Q	State-action function / Critic	–
π	Policy	–
$\mathcal{N}$	Normal distribution	–
μ, σ	Mean/Standard deviation	–
Ω, λ	Actor/Critic parameters	–
γ	Discount factor	–
ρ	Target smoothing coefficient	–
B	Batch size	–
N	Number of critics	–
M	Minimization subset size	–
U	Update-to-data ratio	–
κ	Temperature coefficient	–
$\mathcal{H}$	Target entropy	–

Mathematic notation

L	Loss function	–
η_t, η_s	Temporal/Spatial coefficient	–
τ, T	Training step / Episode length	–
J	Objective	–

Aircraft states and actuators

p, q, r	Pitch/Roll/Yaw rate	[rad/s or deg/s]
V_{TAS}	True airspeed	[m/s]
α, β	Angle of attack/sideslip	[rad or deg]
ϕ, θ, ψ	Roll/Pitch/Yaw angle	[rad or deg]
ϕ_r, θ_r	Roll/Pitch reference	[rad or deg]
ϕ_e, θ_e	Roll/Pitch error	[rad or deg]
h_e	Geometric altitude	[m]
x_e, y_e	Horizontal/Vertical position along Earth	[m]
$\delta_a, \delta_e, \delta_r$	Aileron/Elevator/Rudder deflection	[rad or deg]
T_N	Engine thrust	[N]
ω	Angular frequency	[rad or deg]
t	time / timestep	[s]

[*]MSc. Student, Faculty of Aerospace Engineering, Control and Simulation Division, Delft University of Technology

I. Introduction

The development of aerial vehicle technology has transformed the way in which humans travel, transport goods and defend themselves. Passenger air transport, which suffered a plunge due to the COVID-19 crisis, has recovered from the crisis and reached an all time high as of December 2024 with a forecasted growth of 6.2% in 2025 [1]. Today, pilots reap the rewards of technological innovations by delegating some of, if not all, control tasks to a flight control system. These systems conventionally operate by using linearized aircraft models to decide on actuator control gains to augment or automate many control tasks. However, due to the linearization, the control gains calculated are only valid near fixed operation points, so most systems turn to gain scheduling to obtain the appropriate gains across the vehicle's flight envelope.

Reinforcement learning (RL) is a machine learning technique that offers an alternative to gain scheduling by using agents to learn control tasks. RL refers to a class of machine learning techniques in which agents act and explore their environments through trial and error with the aim of maximizing the notion of discounted cumulative reward [2]. For flight control systems, this notion of reward can be defined to reflect how well a reference is tracked and the actions of the agent the commands given to actuators. This has been shown to be achievable via RL for tracking tasks [3] and for higher level control tasks like autonomous navigation [4]. RL is further partitioned into model based and model free approaches. Model based refers to methods where the agent has access to a partial/complete representation (whether learned or given) of its environment. Model free approaches are those where agents do not have access to an environment and learn solely based on interactions. Model free RL has been successfully used for flight control-relevant applications such as optimal control tracking [5].

RL methods have disadvantages, such as data hunger and safety concerns. Sample efficiency is the term used to quantify the number of steps needed to converge to a good solution. Randomized Ensemble Double Q Learning (REDQ) [6] is a technique used to boost the sample efficiency of actor critic architectures like Soft Actor Critic (SAC), Twin Delayed Deep Deterministic Policy Gradient (TD3) or Deep Deterministic Policy Gradient (DDPG) [7–9] that has seen success across benchmarks. The sample efficiency gains are achieved by introducing an ensemble of critics, minimizing over a random subset of them and increasing the update-to-data (UTD) ratio (which controls how many updates are performed per given batch update step).

The contribution of this paper is to study sample-efficient model free algorithms for flight control use by developing sample-efficient model free attitude tracking controllers using REDQ. It will focus on three control tasks: pitch control, roll control and a combination of the two. The combined roll and pitch attitude tasks will hereafter be referred to as the biaxial control task. Furthermore, the fault tolerance behavior of the more sample-efficient controllers was analyzed under three conditions to investigate potential drawbacks. The experiments were conducted on the CitAST, a high fidelity Cessna Citation 500 model representative of the dynamics of TU Delft's PH-LAB [10].

The paper is structured as follows. Firstly, section II details the background for this paper: a brief recap of RL and Markov Decision Processes (MDPs), an explanation of how REDQ is used on actor critic architectures, how the entropy will be automatically tuned and the policy smoothing approach for future potential use in real systems. Secondly, section III discusses the methodology used for the attitude tracking controllers and the way the experiment was setup and evaluated. Third, section IV shows the results obtained from the experiment. Lastly, section V wraps up with a conclusion.

II. Background

This section will give a brief description of the necessary knowledge that was used in the methodology. First, Reinforcement Learning (RL) and Markov Decision Processes (MDPs) are discussed. Secondly, actor critic architectures are explained with a focus on Randomized Ensemble Double Q Learning (REDQ) [6]. Third, since a Soft Actor Critic (SAC) [7] was the base algorithm chosen, the automatic temperature procedure will be explained. Lastly, to ensure the actuator control signals are apt for real system applications, Conditioning for Action Policy Smoothness (CAPS) [11] is discussed.

A. RL & MDPs

RL is a way of approaching problems using an agent and environment paradigm where the agent navigates sequentially popularized by Sutton and Barto [2]. The agent procedurally selects actions based on a mix of exploration and evaluative feedback without the need to rely on knowledge of what the correct action should be, with the idea that some actions in certain states are more desirable than others and should be more favored if the state were to recur. Figure 1 illustrates the agent environment paradigm where the t subscripts denote a discrete timestep.

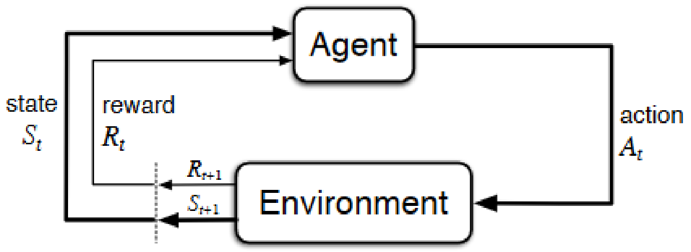


Fig. 1 Agent environment interface [2]

MDPs are a mathematical framework used in reinforcement learning whose formulation can be deterministic or probabilistic. In deterministic MDPs, the environments define the landscape for the agent's learning by means of the state transition function $f : S \times A \longrightarrow S$ and shape the learning via the reward function $r : S \times A \longrightarrow \mathbb{R}$. Agents are said to move through a subset of states S, take actions $a \in A(s)$ depending on a state $s \in S$ based on some policy π. Agents evaluate states using a learned value function such as the state-action value function $Q(s, a)$ (will be used throughout the rest of this paper). Policy optimization methods are those that aim to directly improve an agent's decision-making process and value-based methods aim to improve the agent's evaluation of states and then indirectly arrive at a good policy through selecting actions that maximize value function estimates. In flight control applications where the states and/or action space are continuous, the policies and value functions are often approximated by neural networks.

B. Actor Critic Architectures & REDQ

Actor critic architectures are a hybrid between policy optimization methods and value based methods. They delegate the learning into two coupled roles: the actor which is in charge of the policy and the critic which is in charge of the value function. These two are coupled as the actor's policy takes into account the evaluation estimates of the critic during updates. In continuous state-action space, the policies and value functions tend to be approximated by separate neural networks which are updated based on some loss objective.

SAC is a commonly used actor critic algorithm which uses a stochastic policy that samples from a normal distribution while training. Equation 1 and Equation 2 describe the network update steps for the critic and actor respectively, where $\tilde{a}(s) \sim \pi_\Omega(\cdot|s)$ denotes an action taken by the actor with policy π_Ω constructed with neural network weights Ω. In essence, the critic updates its network parameters λ based on the error relative to a target and the actor updates its network parameters Ω based on the critic evaluations on the actor's previous actions at given states with an entropy penalty to encourage exploration. The weight of the entropy penalty is fixed by hyperparameter k known as the temperature. Two critics are usually used for better performance compared to a single critic [12]. The B representing the batch size is present because these updates are usually performed with batches of state transitions.

$$\nabla_\lambda \frac{1}{|B|} \sum_{(s,a,r,s')\in B} \left(Q_{\lambda_i}(s,a) - y\right)^2 \tag{1}$$

$$\nabla_\Omega \frac{1}{|B|} \sum_{s\in B} \left(\frac{1}{N} \sum_{i=1}^{N} Q_{\lambda_i}(s, \tilde{a}(s)) - \kappa \log \pi_\Omega(\tilde{a}(s)|s) \right), \quad \tilde{a}(s) \sim \pi_\Omega(\cdot|s) \tag{2}$$

REDQ is a technique that builds on actor critic algorithms by layering three extra concepts on regular actor critic architectures to boost their sample efficiency. Firstly, it generalizes the number of critics in the architecture controlled by the hyperparameter N denoting the number of critics. Secondly, it defines a hyperparameter M that is the subset size over which the minimization will be done (used when computing the target to avoid overestimating the values of state-action pairs). The minimization will then be done over a subset M critics which will be chosen randomly from the N total critics. Lastly, a hyperparameter U is defined as the update-to-data ratio (UTD) which sets how often the networks are updated per training step. The REDQ approach can technically be used in any actor critic architectures such as SAC, TD3 or DDPG [8, 9]. Algorithm 1 shows the REDQ implementation used during the experiment.

C. Automatic Entropy Tuning

The weight of the entropy regularization term κ is a hyperparameter that determines how much exploration is encouraged. Naturally, the importance of exploring depends on the stage of learning. Haarnoja et al. [13] proposed a procedure to automatically adjust κ based on the certainty of the agent's actions and a fixed target entropy. The update steps for κ follow Equation 3, where $\mathcal{H}$ is the target entropy determined by the action space dimension. Since automatic tuning is formulated as an optimization problem, it introduces a learning rate hyperparameter. For simplicity, the learning rate for the entropy tuning was the same as for the actor and the critic.

$$\nabla_{\kappa}\mathbb{E}_{a\sim\pi}[-\kappa\log(\tilde{a}(s)|s)-\kappa\mathcal{H}],\quad \mathcal{H}=-\dim(A) \tag{3}$$

D. Policy Smoothing - CAPS

Policies that are capable of successfully performing tasks are not always suitable for use on real-world systems. Often policies are too aggressive in their actions that pose energy, safety and actuator wear concerns. To combat this Mysore et al. [11] demonstrated how penalizing temporal and spacial differences in the actor objective function can yield smooth policies more suitable for real world systems. Temporal loss is defined as the difference in action between two subsequent states. Spacial loss is defined as the difference in action between a state and its surrounding states. These are shown mathematically in Equation 4 and Equation 5.

$$L_t=\eta_t|\tilde{a}(s)-a'(s')| \tag{4}$$

$$L_s=\eta_s|\tilde{a}(s)-\tilde{a}(s+\epsilon)| \tag{5}$$

The actions in a given state are given by the policy $\tilde{a}(s)\sim\pi_{\Omega}(\cdot|s)$. Subsequent actions are also given by the policy using the subsequent state $\tilde{a}(s')\sim\pi_{\Omega}(\cdot|s')$ where $'$ is used to signify subsequent actions or states. The noise added to the state in the spacial loss was sampled from a normal distribution $\epsilon\sim\mathcal{N}(0,0.01)$ for the experiments in this paper. This approach introduces two new hyperparameters to be tuned: η_t and η_s.

III. Methodology

To investigate the sample efficiency and robustness of using REDQ on flight control tasks, three tasks were chosen. These were a pitch tracking task, a roll tracking control task and a task where both these angles were tracked simultaneously termed the biaxial control task. For performance comparison, a standard SAC agent was also implemented. This section discusses how this was accomplished and evaluated.

A. Cessna Citation Model

The control tasks were performed on the CitAST, a non-linear high fidelity model of TU Delft's PH-LAB (Cessna Citation 550) which has been validated using real flight data [10]. The aircraft model consists of twelve states and three actuators given in Equation 6. The relevant sign convention of the states/actuators subset used throughout the experiment is depicted in Figure 2.

$$\begin{aligned} x &= \begin{bmatrix} p & q & r & V_{TAS} & \alpha & \beta & \phi & \theta & \psi & h_e & x_e & y_e \end{bmatrix} \\ u &= \begin{bmatrix} \delta_e & \delta_a & \delta_r & T_{N_1} & T_{N_2} \end{bmatrix} \end{aligned} \tag{6}$$

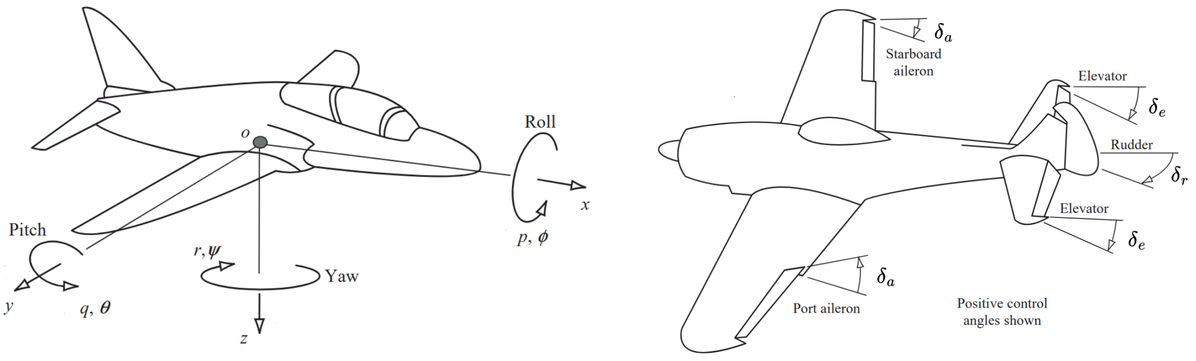


Fig. 2 Relevant aircraft states and actuators sign convention adapted from Cook [14]

The experiment occurred in a trimmed system starting at a 2000 m altitude, 4000 kg aircraft weight, 90 m/s true airspeed and a 0° flight path angle operating point. The system had an already implemented auto-throttle for speed control and a yaw damper. The actuator dynamics were modelled as a saturated first order lag dynamics and ideal sensors were assumed. The simulation update rate of the system was 100 Hz.

B. MDP Formulation

To apply RL theory to the control task, it must first be framed as an MDP. This subsection discusses how each control task was mapped to an MDP formulation. The running example used will be for a REDQ biaxial tracking task since the SAC controllers were constructed similarly.

1. Biaxial control

Figure 3 shows the control diagram for a REDQ biaxial tracking controller. The plant dynamics corresponds to the aforementioned Cessna Citation model, it serves as the environment and gives the reward function (via the reference error). The agent corresponds to the REDQ controller. Training or evaluation was terminated either at the end of an episode or when the agent's actions caused the environment to go out of bounds, resulting in NaN values.

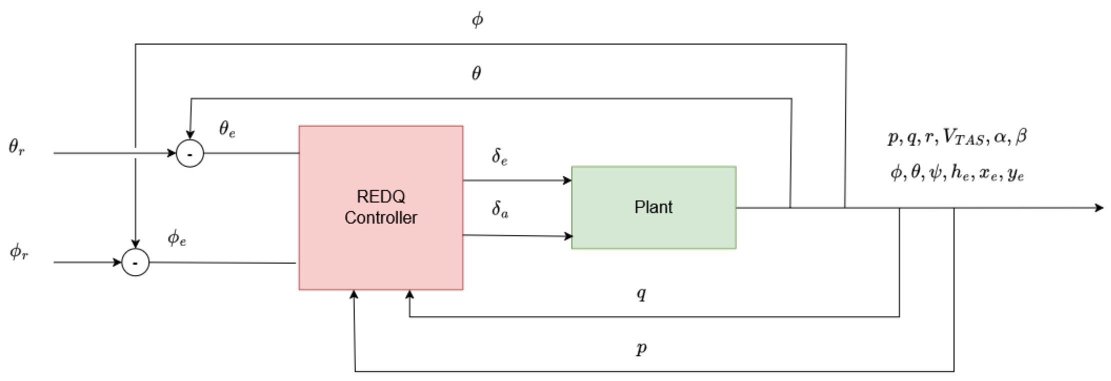


Fig. 3 REDQ controller diagram for biaxial tracking task

The agent states are a four dimensional vector that use the trim condition as starting point. The actions correspond to elevator and aileron commands. The reward is calculated as a weighted sum of the reference errors with respect to each axis where the coefficients were experimentally found based on initial manual changes and looking at the tracking performance on both pitch and roll axes. This is summed up mathematically in Equation 7 and Equation 8. The maximum action was chosen to be constrained within $\delta_e, \delta_a \in [-15°, 15°]$.

$$\begin{aligned} s &= [\theta_e \quad q \quad \phi_e \quad q]^T \\ a &= [\delta_e \quad \delta_a]^T \\ r &= -0.6 \times |\theta_e| - 0.4 \times |\phi_e| \end{aligned} \tag{7}$$

$$\begin{aligned} \theta_e &= \theta_r - \theta \\ \phi_e &= \phi_r - \phi \end{aligned} \tag{8}$$

The REDQ controller or agent consists of an actor network and multiple critic networks, their respective architectures are illustrated in Figure 4. Note that since the number of critics is a hyperparameter for REDQ agents (SAC agents just uses two), there will be more than two critic networks working in parallel but only one actor network.

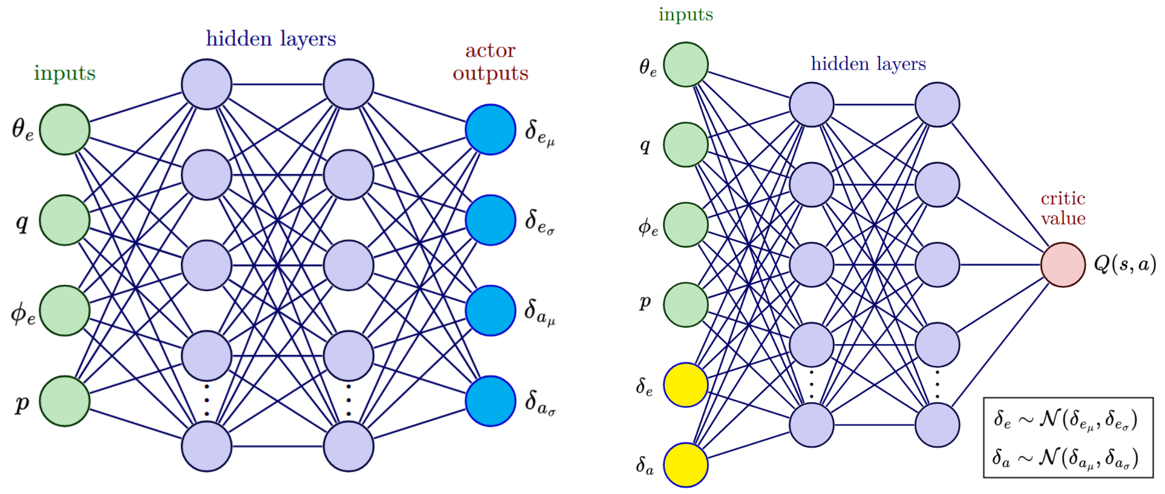


Fig. 4 Network architecture for the actor and the critics

The process of formulating the pitch tracking task and roll tracking task as MDPs is analogous to the process described above but with a reduced state and action space. Mathematically, this is summed up as seen in Equation 9 and Equation 10 for the pitch tracking task and the roll tracking task respectively.

$$\begin{aligned} s &= [\theta_e \quad q]^T \\ a &= [\delta_e] \\ r &= -|\theta_e| \end{aligned} \tag{9}$$

$$\begin{aligned} s &= [\phi_e \quad p]^T \\ a &= [\delta_a] \\ r &= -|\phi_e| \end{aligned} \tag{10}$$

2. Training reference function creation

To train the agent in a generalizable way, the generated reference signal was randomly generated at the beginning of every episode. Equation 11 shows how the pitch reference function was defined, it was designed to prevent unrealistic amplitudes, frequency demands and pitch rate references (its derivative).

$$\theta_r(t) = \sum_k^4 c_k sin(\omega_k t), \quad \sum_k^4 |c_k| = 0.26, \quad 0.05 < \omega_k < 0.2 \text{ [rad/s]} \tag{11}$$

It is a four term sum of sines reference signal where c_k and ω_k are randomly generated with the conditions that the sum of the coefficients be 0.26 and the frequencies bounded. The c_k is allowed to be negative to allow for reference signals that start by pitching down. The coefficient sum condition ensures that under maximum constructive interference the maximum amplitude would be 15° or 0.26 rad. The frequencies are bounded within values that the PH-LAB can respond to (found experimentally by probing the model with difference frequencies). Since the sum of different frequencies can lead to unexpected effective frequencies along it's trajectory, a lowpass Butterworth filter with corner frequency 0.2 rad/s is used on the signal to ensure that the model can always respond to the reference signal. The roll reference signal $\phi_r(t)$ was constructed identically.

C. Algorithm Configuration

The version of REDQ SAC that was implemented for the controllers can be seen in Algorithm 1. Most notably when compared with the reference REDQ algorithm, there is no hyperparameter defining the subset size over which to minimize the critics. Additionally, automatic entropy tuning is added and the actor loss function has the added CAPS regularization terms.

Algorithm 1 REDQ SAC Algorithm based on Chen et al. [6]

```
1: Initialise policy parameters $\Omega$, $N$ Q-function parameters $\lambda_i$, $i = 1, \ldots, N$, empty replay buffer $\mathtt{D}$.
2: Set target parameters $\lambda_{\text{targ},i} \leftarrow \lambda_i$, for $i = 1, 2, \ldots, N$
3: loop
4:     Take one action $a_t \sim \pi_\Omega(\cdot|s_t)$. Observe reward $r_t$, new state $s_{t+1}$
5:     Add data to buffer: $\mathtt{D} \leftarrow \mathtt{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
6:     for $U$ updates do do
7:         Sample a mini-batch $B = \{(s, a, r, s')\}$ from $\mathtt{D}$
8:         Compute the Q target $y$ (same for all of the $N$ Q-functions):
               $y = r + \gamma \left( \min_{i \in N} Q_{\lambda_{\text{targ},i}}(s', a') - \kappa \log \pi_\Omega(a'|s') \right), \quad a' \sim \pi_\Omega(\cdot|s')$
9:         for $i = 1, \ldots, N$ do do
10:            Update $\lambda_i$ with gradient descent using
                   $\nabla_\lambda \frac{1}{|B|} \sum_{(s,a,r,s') \in B} \left( Q_{\lambda_i}(s, a) - y \right)^2$
11:        end for
12:        Update target networks with $\lambda_{\text{targ},i} \leftarrow \rho \lambda_i + (1 - \rho) \lambda_{\text{targ},i}$
13:    end for
14:    Update policy parameters $\Omega$ with gradient ascent using
           $\nabla_\Omega \frac{1}{|B|} \sum_{s \in B} \left( \frac{1}{N} \sum_{i=1}^{N} Q_{\lambda_i}(s, \tilde{a}(s)) - \kappa \log \pi_\Omega(\tilde{a}(s)|s) - \eta_t |\tilde{a}(s) - a'(s')| - \eta_s |\tilde{a}(s) - \tilde{a}(s + \epsilon)| \right),$
           $\tilde{a}(s) \sim \pi_\Omega(\cdot|s), \quad a'(s) \sim \pi_\Omega(\cdot|s'), \quad \epsilon \sim \mathcal{N}(0, 0.01)$
15:    Update temperature $\kappa$ using gradient descent using
           $\nabla_\kappa \mathbb{E}_{a \sim \pi} [-\kappa \log(\tilde{a}(s)|s) - \kappa \mathcal{H}], \quad \mathcal{H} = -\dim(A)$
16: end loop
```

The rationale for discarding the critic minimization subset size M as a hyperparameter for REDQ was done because the number of critics was kept low due to computational considerations. As such, any minimization over a subset of the critics would not have as big an impact as if there were more critics. Additionally, Hiraoka et al. [15] demonstrated that minimizing over the full number of critics yields sample efficient performance in conjunction with the use of dropout on several benchmarks.

Given how sensitive the performance of agents can be to hyperparameters, a study was conducted to find a good selection of hyperparameters for the SAC agent pitch tracking task using Optuna's TPE sampling [16] with objective: maximizing the asymptotic return. These same hyperparameters were used for the roll tracking task and biaxial tracking task as they were found to work well for those too. A separate hyperparameter study for REDQ agents was performed but ultimately not used. This is because the objective for the hyperparameter studies was tied to the average asymptotic return instead of the sample efficiency. Instead, to isolate the impact REDQ hyperparameters had on sample efficiency, two agents were proposed RED3Q and RED5Q. These agents had a matching number of UTD and number of critics set by the number in the name. RED3Q was proposed as a minimal incremental step to SAC, only increasing the number of critics by one and tripling the UTD ratio. RED5Q constituted a larger step with respect to SAC, adding three more critics and quintupling the UTD ratio. A summary of the hyperparameters used for the SAC and REDQ SAC agents can be seen in Table 1 and Table 2 respectively.

Table 1 Hyperparameters SAC agents

Parameter	Value
Optimizer	Adam [17]
Nonlinearity	ReLU
# hidden layers	2
# of units per hidden	64
Target entropy	$-\dim(A)$
Spacial smoothness coeff.[1]	5 | 30
Temporal smoothness coeff.[1]	15 | 60
Learning rate	6.3×10^{-4}
Discount factor	0.99
Target smoothing coeff.	0.076
Batch size	64
Buffer size	50k

[1]Left was for pitch or roll tasks. Right for biaxial task.

Table 2 Hyperparameters REDQ agents

Parameter	Value
Optimizer	Adam [17]
Nonlinearity	ReLU
# hidden layers	2
# of units per hidden	64
Target entropy	$-\dim(A)$
Spacial smoothness coeff.[1]	5 | 30
Temporal smoothness coeff.[1]	15 | 60
Learning rate	6.3×10^{-4}
Discount factor	0.99
Target smoothing coeff.	0.076
Batch size	64
Buffer size	50k
# of critics[2]	3 | 5
UTD ratio[2]	3 | 5

[1]Left was for pitch or roll tasks. Right for biaxial task.
[2]For RED3Q and RED5Q respectively.

The choice of keeping the base hyperparameters the same was done in order to attribute the performance differences to the REDQ hyperparameters as much as possible. The reason both RED3Q and RED5Q were used for evaluation was to get a wider view of how these hyperparameters affect performance. Note that the hyperparameter study was only conducted on the third group of hyperparameters in the table, the parameters for the first group were based on Dally [18] and the CAPS hyperparameters were found experimentally.

D. Experiment Setup

This subsection will discuss how the agents were trained and evaluated. Evaluation involved performing statistical analysis on the learning curves, as well as performing a fault tolerance analysis.

1. Training

The training was divided into 18 different blocks and used randomly generated attitude reference signals. The 18 blocks are a result of the combination of the three control tasks, the three agents (SAC, RED3Q, RED5Q) and two training scopes: a short term analysis with more granularity in the beginning where most of the learning happens to compare sample efficiency and a long term analysis for a lower resolution picture of the asymptotic performance when the learning has converged. Table 3 details the scope specifications per task for each agent. The short scope was used for the learning curves as this encompassed the first 12,500 steps where most of the learning took place. The network weights at the end of the 25,000 step long scope were used to compare the fault tolerance between the agents when the learning had converged. Note that 30 agents are trained to get a better estimate of the performance of a given agent per task per scope.

Table 3 Scope specification for agent training per control task

Scope	Num. of agents	Num. of episodes	Ep. length [steps]	Resolution [steps]
Long	30	10	2500	250
Short	30	5	2500	50

To monitor the agent's tracking performance, a function is used to evaluate the agent's performance at every checkpoint. When evaluating, the mean action of the agent is used (as opposed to the stochastic action used during training). The function used to evaluate the performance of the agent at every snapshot is given in Equation 12 (the roll references are constructed identically but with its respective initial trim offset). The tracking performance was evaluated with respect to the reference for twenty seconds with a system update rate of 100 Hz, which results in a 2000 step evaluation episode.

$$\theta_{r_e}(t) = 0.12sin(0.4t) + \theta_0, \quad t \in [0, 20] \tag{12}$$

The learning curves were constructed by plotting the evaluation performance against the number of steps the agent trained. They show the mean and 95% confidence intervals of all snapshots across the 30 agents per given task and scope. Although all 30 agents eventually converged to good solutions, a few of them were not used during the analysis as some training episodes were terminated due to going out of bounds, resulting in NaN values.

2. Statistical analysis

To fairly compare performance between SAC and REDQ agents, two benchmark performance objectives were set as seen in Equation 13 and Equation 14 with τ representing the number of steps. The first threshold of -5000 was chosen as at this point agents already have a decent policy. For some intuition, given the evaluation episodes were 2000 steps long a reward of -5000 would mean on average the agent was off reference by $|\frac{5000}{2000}| = |2.5°|$ since the reward function was directly tied to the reference error. Similar reasoning was used for the -2000 threshold which meant being off reference by $|1°|$ on average, typical of fully trained agent performance.

$$J_{5000} = \min|\tau| \text{ s.t. } r_\mu(\tau) > -5000 \tag{13}$$

$$J_{2000} = \min|\tau| \text{ s.t. } r_\mu(\tau) > -2000 \tag{14}$$

To make meaningful observations about the learning tasks, a statistical analysis of the agents was performed. For this, two metrics were used. Firstly, a Welch t-test was used to determine whether the differences in the learning curves were statistically significant. The associated p values were then used in combination with the learning curves to determine which agent performs best. Secondly, to analyse whether the differences are practically meaningful, the mean error (ME) relative to the reference per evaluation is logged. These are calculated as seen in Equation 15 where T is the length of the evaluation.

$$\begin{aligned} \mathrm{ME}_\theta &= \frac{1}{T}\sum_{t=1}^{T}\theta_e \\ \mathrm{ME}_\phi &= \frac{1}{T}\sum_{t=1}^{T}\phi_e \end{aligned} \tag{15}$$

3. Fault tolerance analysis

To test the robustness of the agents, their fault tolerance behavior was analyzed under three conditions. The nominal condition consisted of no faults introduced and evaluated the agent performance on a sinusoid for 2 minutes with the aim of seeing how agents perform over longer time horizons. The quarter actuator efficiency condition consisted of multiplying the actuator command by $\frac{1}{4}$ and evaluating its performance on randomly generated reference signals with diminished actuator control. The purpose of this scenario was to test whether the agents were robust to large changes in actuator effectiveness. The actuator jolt condition pushed the actuator to a maximum deflection for a second in each direction with the aim of testing if the agent was able to recover from sudden large disturbances. The mean error for each agent was recorded for every condition and presented in a box plot showing their interquartile range (IQR).

The mathematical formulation for each of the conditions are detailed hereafter. For brevity, it is exemplified as if it was for a pitch control task. The roll and biaxial task were constructed in the same manner so their definitions are analogous. For the nominal condition, the reference function is defined as seen in Equation 16 and the actuator will follow Equation 17. The μ subscript is used to signify that the mean action is used for evaluation as opposed to the stochastic one used during training.

$$\theta_r(t) = 0.12sin(0.4t) + \theta_0, \quad t \in [0, 120] \tag{16}$$

$$\delta_e = a_\mu(s) \sim \pi_\Omega(\cdot|s) \tag{17}$$

For the quarter actuator efficiency condition, the randomly generated reference was constructed, as previously discussed, using Equation 11 for $t \in [0, 60]$ and the actuator will follow Equation 18.

$$\delta_e = \frac{1}{4} \times a_\mu(s) \sim \pi_\Omega(\cdot|s) \tag{18}$$

For the actuator jolt condition, the definitions for reference and actions are given in Equation 19 and Equation 20.

$$\theta_r(t) = 0, \quad t \in [0, 20] \tag{19}$$

$$\delta_e(t) = \begin{cases} 15^\circ & 5 \leq t \leq 6 \\ -15^\circ & 10 \leq t \leq 11 \\ a_\mu(s) \sim \pi_\Omega(\cdot|s) & \text{otherwise} \end{cases} \tag{20}$$

IV. Results and Discussion

As discussed in the methodology, the results will be divided in two sections. The results relating to sample efficiency and learning curves are given in subsection IV.A and the fault tolerance analysis will be discussed in subsection IV.B.

A. Sample Efficiency Analysis

The learning curve for the pitch tracking task and the SAC vs. RED3Q Welch t-test are shown in Figure 5. The t-test was conducted between SAC and RED3Q as both REDQ agents showed similar initial performance, with RED3Q showing better asymptotic stability. After initial randomness in early learning, both REDQ agents learn faster as seen in their fast rise towards the settling region in approximately 1000 steps, with the differences between SAC and RED3Q seen being statistically significant (p value < 0.05). After approximately 2000 steps, SAC agents reached comparable performance levels, at which point the differences between RED3Q and SAC were no longer statistically significant.

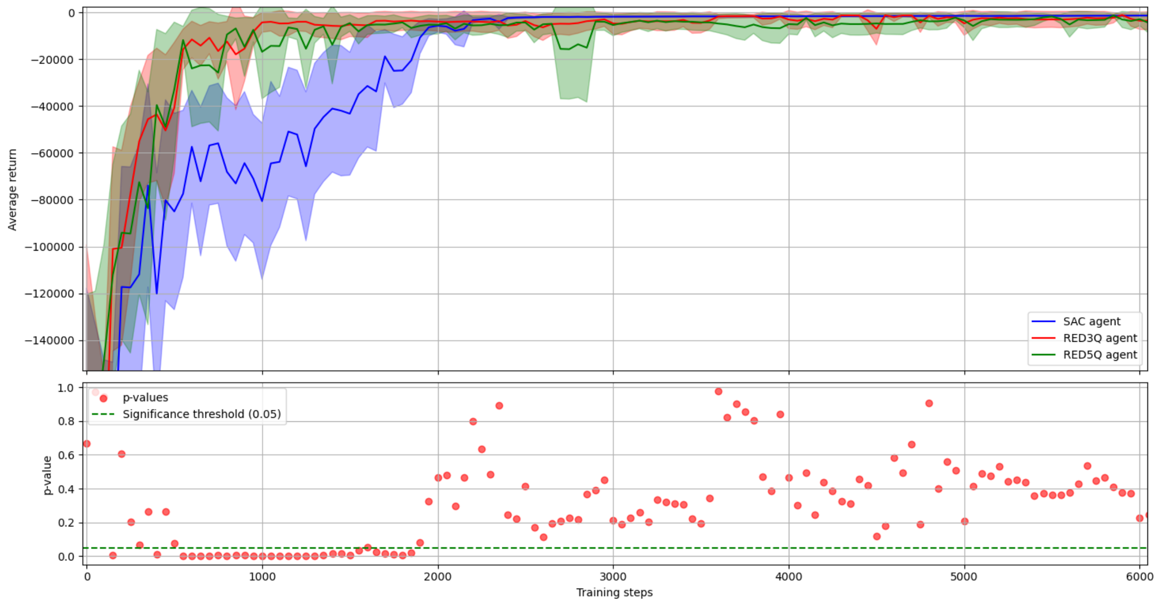


Fig. 5 Top: Pitch tracking learning curves for SAC, RED3Q, and RED5Q (each with N = 30) Bottom: Unpaired t-test p-values comparing SAC vs. RED3Q (both with N = 30)

SAC exhibits more stable asymptotic stability, as the learning curve trajectory remains more leveled after reaching the settling region. In contrast, both RED3Q and RED5Q show higher variance in performance in the settling region.

This pattern is consistent with the benchmark results presented in Table 4, where the number of steps for J_{5000} is improved by $\sim$ 50% and $\sim$ 25% for RED3Q and RED5Q respectively when compared to SAC. SAC performs the best in the number of steps for J_{2000} with a $\sim$ 30% and $\sim$ 50% improvement over RED3Q and RED5Q respectively.

Table 4 Pitch control benchmark performance comparison

Agent	J_{5000} (steps)	J_{2000} (steps)
SAC	2200	2600
RED3Q	1000	3600
RED5Q	1500	5500

*J is the number of steps until subscript level mean performance is achieved.

The roll control learning curve and SAC vs. RED3Q Welch t-test shown in Figure 6 showed characteristics similar to those discussed for pitch. After the initial randomness, the REDQ agents quickly rose to near settling region within 800 steps, demonstrating better sample efficiency. Performance differences were statistically significant between steps 300-1400 and then towards the tail of the learning curves between 4300-5800 steps. RED5Q once again demonstrated volatile asymptotic stability.

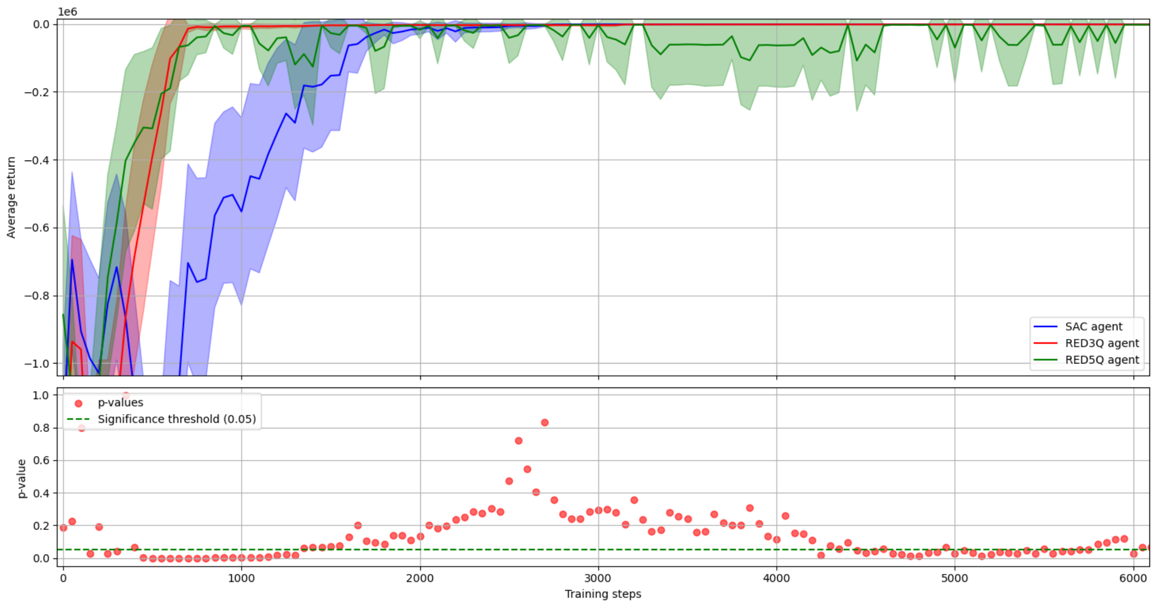


Fig. 6 Top: Roll tracking learning curves for SAC (N = 30), RED3Q (N = 27), and RED5Q (N = 30) Bottom: Unpaired t-test p-values comparing SAC (N = 30) vs. RED3Q (N = 27)

The first statistically significant signal is due to the gains in sample efficiency. The second signal was due to the SAC agents demonstrating better asymptotic performance during this part of the settling region. Although statistically significant, the second signal was not numerically meaningful with a difference in the average return of $\sim$ 100.

The benchmark comparison for the roll control task is given Table 5. In terms of agent performance, it tracked with the performance shown on the pitch task. Given that both pitch and roll benchmarks were met in approximately the same number of steps, this indicates that longitudinal and asymmetric dynamics present comparable levels of learning complexity for the agents.

Table 5 Roll control benchmark performance comparison

Agent	J_{5000} (steps)	J_{2000} (steps)
SAC	2550	2750
RED3Q	1400	3150
RED5Q	1450	6400

*J is the number of steps until subscript level mean performance is achieved.

The biaxial learning curve and SAC vs. RED3Q Welch t-test are shown in Figure 7. The climb to near settling region for all agents occurs within 2000 steps, with REDQ agents showing higher return within this period. Compared to previous control tasks, the biaxial task showed higher volatility at the beginning of learning and a higher number of steps required to learn the tasks. The gains in performance for the biaxial task were less pronounced, but still significant statistically and meaningful in terms of numerical magnitude.

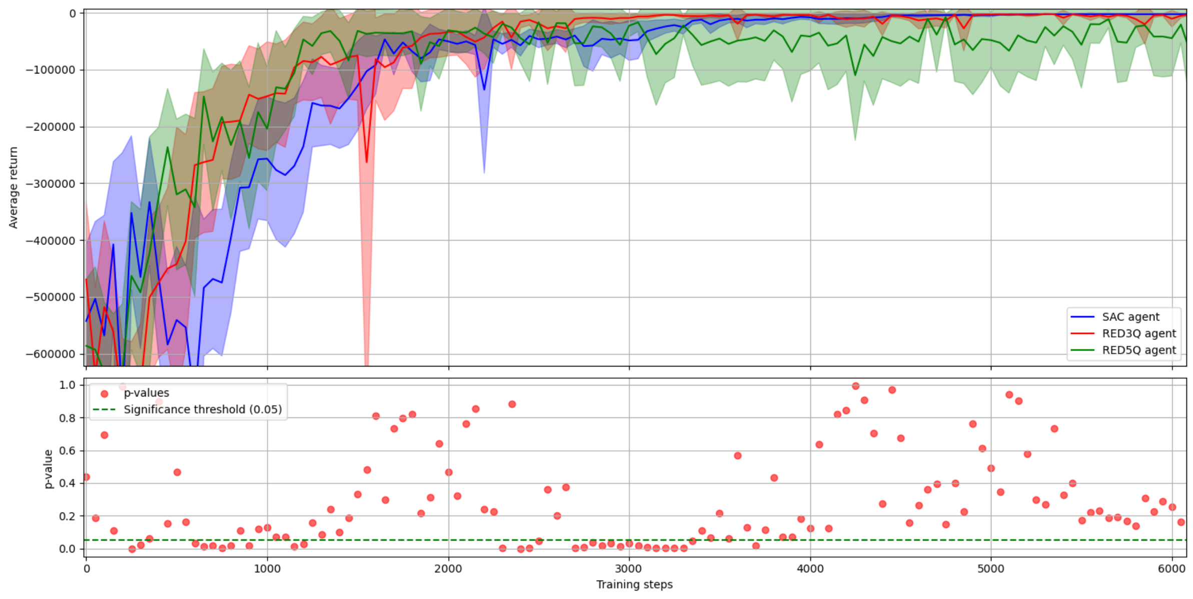


Fig. 7 Top: Biaxial tracking learning curves for SAC, RED3Q, and RED5Q (each with N = 30) Bottom: Unpaired t-test p-values comparing SAC vs. RED3Q (both with N = 30)

The first statistically significant signal occurred between 500-1000 steps, during which REDQ agents consistently outperformed SAC, as agents learned faster after the initial randomness. The second signal took place between 2300-3400 steps, where RED3Q climbed to asymptotic performance faster after the learning had begun to level. Between these two intervals, RED3Q exhibited a temporary drop in performance around 1500 steps. This local mismatch with the general trend is attributed to outliers causing a large drop in relative performance.

The benchmark comparison for the biaxial control task is given in Table 6. The thresholds were all met after a longer number of steps compared to the pitch and roll control tasks. This signals a higher learning complexity and was expected as the agents have to identify which actuators affect the longitudinal or asymmetric states. The weak dynamic coupling between longitudinal and asymmetric states likely played a role too. RED3Q agents took 25% less steps to achieve J_{5000} than SAC but took double the steps to achieve J_{2000}. This is consistent with the findings of previous tasks in which REDQ agents outperformed early on but had more volatile asymptotic stability. RED5Q agents performed poorly across the board, with J_{2000} never being met within the training scope. This is possibly due to convergence to a local minimum in this more complex task given its higher UTD ratio.

Table 6 Biaxial control benchmark performance comparison

Agent	J_{5000} (steps)	J_{2000} (steps)
SAC	4400	6600
RED3Q	3150	11150
RED5Q	7600	-[1]

*J is the number of steps until subscript level mean performance is achieved.
[1]RED5Q did not achieve a mean performance of -2000 within the short training scope.

All agents were able to learn all three tracking tasks to a degree within the training with a 100% success rate as all policies eventually converged, although with varying volatility in asymptotic stability. For pitch and roll tasks, the tasks were learned to the J_{5000} standard within 2600 steps and to the J_{2000} standard within 6400 steps. The biaxial task showed roughly a doubling in the number of steps required to achieve benchmark performance, with J_{5000} achieved in 4400 steps (excluding RED5Q) and J_{2000} after 11200 steps (once again excluding RED5Q).

The general trend of quicker learning at the expense of stability down the line is indicative of a trade-off between sample efficiency and worsened asymptotic performance likely due to overfitting. Overfitting being the cause of the worsened asymptotic performance is further supported by RED5Q showing a higher instability in the settling region compared to RED3Q. One possible interpretation is that, despite a higher number of critics that aim to prevent overestimation of states, occasional overfitting occurs as a result of the higher UTD ratio once in the settling region. RED5Q agents did not demonstrate faster initial learning either when compared to RED3Q, pointing to the idea that, at least for tasks in which agents can learn quickly, there is a limit to the gains in sample efficiency from an increased number of critics and UTD ratio. Intuitively, this makes sense as there must be a limit to the amount of information agents can extract from data samples through simply increasing these two parameters.

B. Fault Tolerance Analysis

The nominal condition performance for both pitch and roll is shown in Figure 8. In the pitch plot, all agent error values within the IQR were below 1°. Excluding outliers, REDQ agents outperformed SAC agents, with RED3Q performing slightly better than RED5Q. Furthermore, SAC had no outliers, signaling all 30 agents had accurate tracking performance. In contrast, both RED3Q and RED5Q showed outliers, with the worst-performing RED3Q outlier having an error an order of magnitude greater than the second worst.

The roll plot demonstrates similar behavior to that of pitch, all IQRs were below 1° and the REDQ agents performed better with respect to IQR to SAC. The biggest difference between the pitch and roll plots is that the SAC agent did have outliers, whereas the number of outliers for REDQ agents remained roughly the same. The behavior observed across these two tasks is not unexpected, as the nominal condition is closest to the evaluation metric used to generate the learning curves. Since all learning curves eventually converged, it makes sense for the performance of all agents to be very close to the reference after being trained over a long scope.

These results show that the agents were able to generalize beyond the training episode duration, at least when moving from 20-25 seconds per episode to 120 seconds. Longer episode durations could be problematic, as some states like the yaw and the altitude remain uncontrolled and will slowly drift the further the episode goes past the initial trim operating point at which the simulation starts. This can be compensated by introducing altitude and yaw control, possibly through the use of additional RL agents to enable full attitude and altitude control.

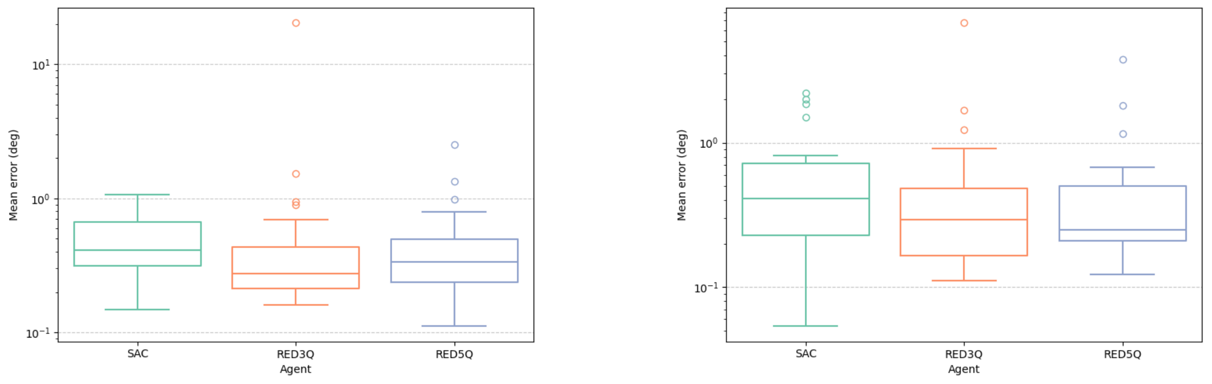


Fig. 8 Nominal condition task: Pitch (left) SAC, RED3Q, RED5Q (N = 30 for each) Roll (right) SAC (N = 30), RED3Q (N = 27), RED5Q (N = 30)

The nominal condition performance for the biaxial controller is shown in Figure 9, where the error per axis is given for each agent. Again, the performance of all agents was very close to a 1° error on both the pitch and roll axes. The main difference observed was a wider IQR compared to the individual pitch and roll tasks, likely due to the increased complexity of the biaxial task. Performance was generally consistent across all agents, with SAC and RED3Q exhibiting lower errors in roll than in pitch. Conversely, RED5Q demonstrated better performance along the pitch axis, which is what was originally expected, as the reward function for the biaxial task penalizes pitch error more heavily than roll error. Evidently, the penalization choice did not affect the final error performance between axes for the nominal condition. Overall, the biaxial agents were also able to track references in both axes with similar accuracy even when extended beyond the training episode duration.

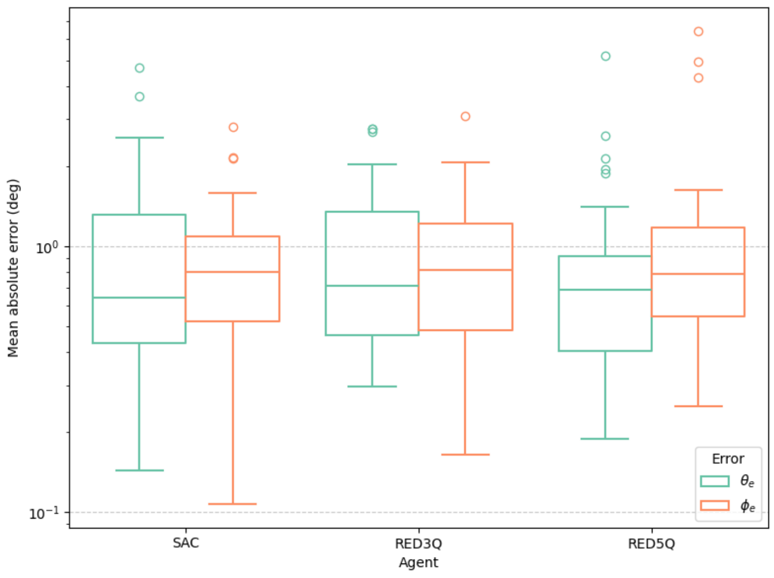


Fig. 9 Nominal condition task: Biaxial SAC (N = 30), RED3Q (N = 29), RED5Q (N = 30)

The quarter elevator effectiveness condition performance for the pitch and roll is shown in Figure 10. Performance wise it was very similar to what was seen under nominal conditions, with all agents having an IQR mean error below 1°. REDQ agents once again demonstrated better performance than SAC agents. In general, the results suggest that all agents are capable of executing control tasks at a level comparable to nominal performance, even with only 25% actuator effectiveness. This suggests that the pitch and roll policies policies learned by the agents are not minimal in action, as even with diminished actuator control the agents are still able to effectively control the system.

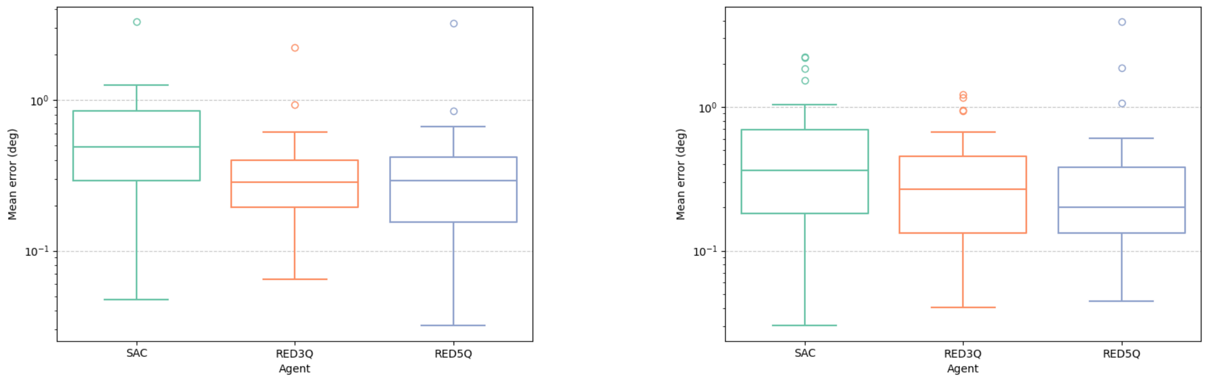


Fig. 10 Quarter actuator effectiveness condition task: Pitch (left) SAC, RED3Q, RED5Q (N = 30 for each) Roll (right) SAC (N = 30), RED3Q (N = 27), RED5Q (N = 30)

The quarter elevator effectiveness condition performance for the biaxial controller is shown in Figure 11. The IQR performance was similar to that of the nominal biaxial condition, staying within the 1° error range. This demonstrates that for the biaxial agents, the policies learned were also not minimal. The most notable distinguishing feature of this plot is the RED5Q outlier performance, which shows mean errors ranging from 10° up to almost 100°. This is believed to result from two factors. First, the aircraft's ability to deviate further in roll, as indicated by the large deviations occurring primarily along this axis. Second, overfitting to nominal conditions due to the higher UTD ratio during training, as evidenced by the RED5Q agent exhibiting the largest magnitude of outlier errors. Overall, nearly all agents mirrored their performance under nominal conditions, indicating that the trained agents are robust to changes in actuator effectiveness. The exception was RED5Q, whose outliers demonstrated significantly worse performance.

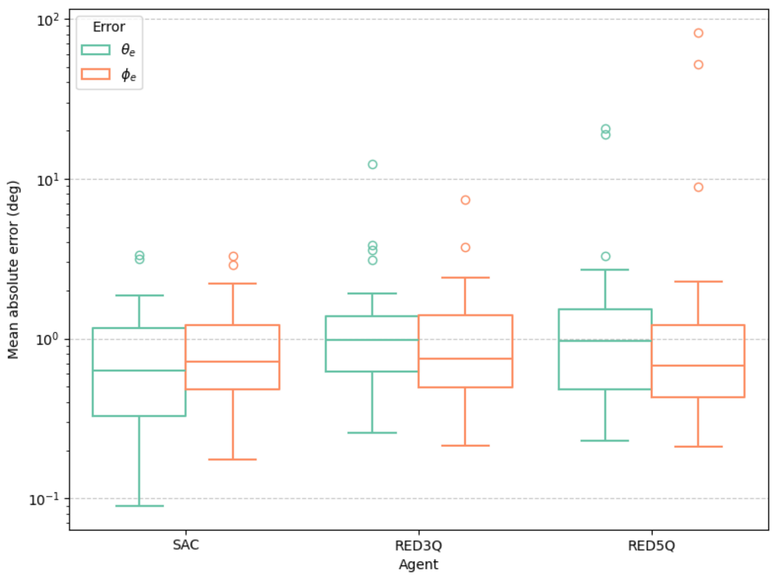


Fig. 11 Quarter elevator effectiveness condition task: Biaxial SAC (N = 30), RED3Q (N = 29), RED5Q (N = 30)

The actuator jolt condition performance for both pitch and roll are shown in Figure 12. As seen in the figure, SAC and RED3Q performed comparably in terms of IQR with SAC showing better overall performance due to the presence of no outliers. The results suggest that SAC agents generalized better to this fault condition than the other two types of agents. Furthermore, the roll performance was worse for the REDQ agents, with the third quartile for both showing a roll error higher than 10°. This once again highlights the combination of higher errors caused by the possibility of larger deviations in roll and overfitting. Of all of the conditions, this one was expected to generate the highest number of outliers, as it purposefully put the agents in states far away from nominal conditions. The fact that SAC agents were able to perform the best is attributed to the slower initial learning, where SAC agents had more time to explore these non-nominal states and sharpen their policies outside of the nominal region. In contrast, REDQ agents showed poor robustness to large deflections, attributed to their faster initial learning and possibly overfitting. Overfitting is once again suspected, as RED5Q performed worse relative to RED3Q in terms of mean error IQR.

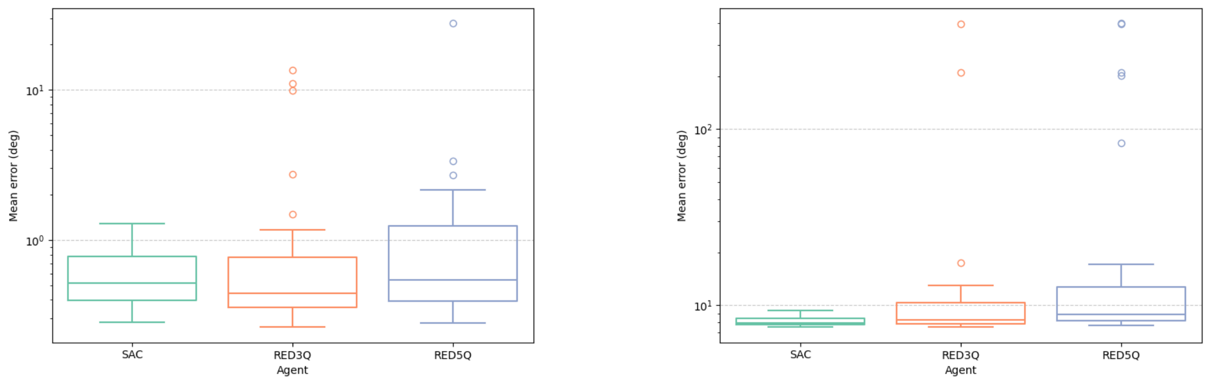


Fig. 12 Actuator jolt condition task: Pitch (left) SAC, RED3Q, RED5Q (N = 30 for each) Roll (right) SAC (N = 30), RED3Q (N = 27), RED5Q (N = 30)

The actuator jolt condition for the biaxial controller is shown in Figure 13. Among the tested agents, SAC exhibited the best IQR performance across both the pitch and roll axes. The outlier mean error performance in roll across all agents was very large, with even SAC showing 100° mean error outliers. This supports once again that roll is the more manoeuvrable axis. RED3Q showed comparable performance along the pitch axis, but with deteriorated roll axis performance. RED5Q performed the worst by a significant margin, with median errors exceeding 10° on both axes. In particular, the mean error on RED5Q's roll axis was concerning, with mean errors on the third quartile approaching 80°. This suggests that RED5Q struggles to generalize to large disturbances, likely a result from overfitting to nominal conditions as a result of the high UTD ratio once learning reaches the settling region. The results are consistent with trends observed in the pitch and roll actuator jolt fault scenarios, where the less sample-efficient SAC agents demonstrated a greater ability to generalize under large disturbances.

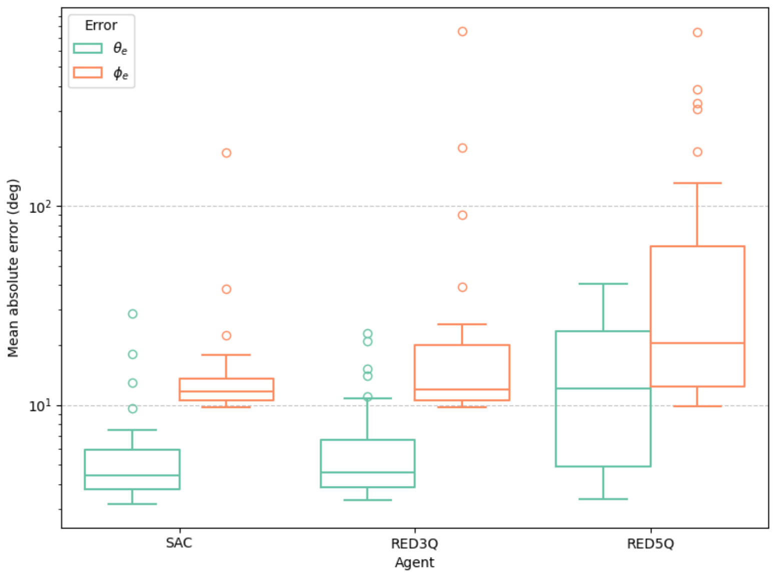


Fig. 13 Actuator jolt condition task: Biaxial SAC (N = 30), RED3Q (N = 29), RED5Q (N = 30)

No condition evaluations resulted in terminations from going out of bounds. However, this should not be interpreted as a measure of agent success. Some cases showed undesirable behavior. In particular, outlier performance or those involving RED5Q under actuator jolt faults demonstrated performance that would be considered unacceptable for deployment in a real aircraft. For the nominal and quarter elevator effectiveness conditions, all agents generally demonstrated good tracking performance with low mean errors. In contrast, the biaxial actuator jolt condition resulted in significantly higher mean errors for all agents. This fault scenario proved to be the most challenging, with all agents showing their worst performance relative to the other fault conditions discussed. This outcome was expected, as the biaxial actuator jolt introduces large disturbances and pushes the controllers into non-nominal states.

All agents were trained for the same number of environment steps, but it is likely that REDQ agents would benefit more from early stopping due to the potential for higher UTD ratios to cause overfitting once learning converges. This demonstrates the challenge of defining a fair comparison point for fault tolerance for the different agents. The approach taken focused on asymptotic performance after training convergence, which simplifies the comparison process and gives an insight into the long term stability and robustness of the learned policies.

V. Conclusion

Three controllers are developed for pitch, roll and biaxial attitude tracking tasks on a high fidelity non-linear Cessna Citation model. Two controllers use a REDQ SAC implementation, while the third uses a standard SAC architecture for comparison purposes. All agents converge to a solution with average accurate tracking convergence (error $< 1°$) occurring within 5,500 training steps for pitch, 6,400 for roll (excluding RED5Q) and 11,500 for biaxial control (excluding RED5Q again). Similarity between number of steps to convergence between pitch and roll indicates similar learning complexity for the agents for longitudinal and asymmetric dynamics. On the contrary, the biaxial task requiring roughly double the number of steps until accurate tracking convergence is attributed to increased task complexity. REDQ agents show statistically significant improvements in sample efficiency during initial learning across all tasks. A trade-off between sample efficiency and asymptotic learning stability is observed in the learning curves, it is attributed to overfitting stemming from a higher update-to-data ratio once within the settling region. This is further supported by RED5Q showing worse asymptotic learning stability than RED3Q.

After training is left to converge over 25,000 steps, a fault tolerance analysis is conducted. Results show comparable performance in nominal conditions across all agents with a higher number of outliers being associated with REDQ agents, particularly RED5Q. This is suspected to be due to the high update-to-data ratio causing overfitting after reaching the learning settling region. The quarter actuator effectiveness condition results show comparable performance with nominal conditions and demonstrate agents are able to track references with diminished actuator control. This is believed to be a result of converged policies being non-minimal in terms of action. The fault scenario associated with the highest magnitude errors and highest number of outliers is the actuator jolt condition, particularly on the biaxial task. Surprisingly SAC agents show no outliers in the pitch and roll tasks signaling strong generalizability. Outliers are present for SAC agents in the biaxial task but are much more pronounced in REDQ agents with RED5Q third quartile mean error being almost 80°. This is attributed to the slower learning of SAC actually being advantageous as it spends more time under non-nominal conditions during training.

This research demonstrates that model free offline trained reinforcement learning agents are able to learn attitude control tasks within a low number of training steps. Furthermore, REDQ can be used to further improve the initial sample efficiency in attitude flight control tasks, but at the expense of potential learning stability and robustness drawbacks should the learning not be halted timely. REDQ agents in this paper use matching number of critics and update-to-data ratios, leaving how these hyperparameters affect the learning individually at large and is recommended for future research. Additionally, the focus of this research was on the SAC variant of REDQ, the question of how REDQ performs in other actor critic architectures in the tasks proposed remains open. Given the small number of training steps necessary to learn the control tasks presented in this paper, future work should examine whether strategies similar to REDQ could enable agents to learn entirely online. Greedier, more deterministic, methods that use a warm start or online-offline hybrids architectures are promising candidates.

References

[1] International Air Transport Association (IATA), "Global Outlook for Air Transport: A World with Lower Oil Prices?" Tech. rep., International Air Transport Association, December 2024. URL `https://www.iata.org/en/iata-repository/publications/economic-reports/global-outlook-for-air-transport-december-2024/`.

[2] Sutton, R. S., and Barto, A. G., *Reinforcement learning: An introduction*, MIT press, 2018.

[3] Konatala, R., Van Kampen, E.-J., and Looye, G., "Reinforcement Learning based Online Adaptive Flight Control for the Cessna Citation II (PH-LAB) Aircraft," *AIAA Scitech 2021 Forum*, 2021, p. 0883.

[4] Wang, C., Wang, J., Shen, Y., and Zhang, X., "Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach," *IEEE Transactions on Vehicular Technology*, Vol. 68, No. 3, 2019, pp. 2124–2136. https://doi.org/10.1109/TVT.2018.2890773.

[5] Luo, B., Liu, D., Huang, T., and Wang, D., "Model-Free Optimal Tracking Control via Critic-Only Q-Learning," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 27, No. 10, 2016, pp. 2134–2144. https://doi.org/10.1109/TNNLS.2016.2585520.

[6] Chen, X., Wang, C., Zhou, Z., and Ross, K., "Randomized Ensembled Double Q-Learning: Learning fast without a model," 2021. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85150281117&partnerID=40&md5=0194d3d5c7b87619a22fa6799ebf4d3c.

[7] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," , 2018. URL https://arxiv.org/abs/1801.01290.

[8] Fujimoto, S., van Hoof, H., and Meger, D., "Addressing Function Approximation Error in Actor-Critic Methods," , 2018. URL https://arxiv.org/abs/1802.09477.

[9] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," , 2019. URL https://arxiv.org/abs/1509.02971.

[10] van den Hoek, M. A., de Visser, C. C., and Pool, D. M., "Identification of a Cessna Citation II Model Based on Flight Test Data," *Advances in Aerospace Guidance, Navigation and Control*, Springer International Publishing, 2018, pp. 259–277.

[11] Mysore, S., Mabsout, B., Mancuso, R., and Saenko, K., "Regularizing Action Policies for Smooth Control with Reinforcement Learning," , 2021. URL https://arxiv.org/abs/2012.06644.

[12] van Hasselt, H., Guez, A., and Silver, D., "Deep Reinforcement Learning with Double Q-learning," , 2015. URL https://arxiv.org/abs/1509.06461.

[13] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S., "Soft Actor-Critic Algorithms and Applications," , 2019. URL https://arxiv.org/abs/1812.05905.

[14] Cook, M. V., *Flight Dynamics Principles*, Elsevier Ltd., 2013.

[15] Hiraoka, T., Imagawa, T., Hashimoto, T., Onishi, T., and Tsuruoka, Y., "Dropout Q-Functions for Doubly Efficient Reinforcement Learning," , 2022. URL https://arxiv.org/abs/2110.02034.

[16] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B., "Algorithms for Hyper-Parameter Optimization," *Advances in Neural Information Processing Systems*, Vol. 24, edited by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Curran Associates, Inc., 2011, p. 4. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[17] Kingma, D. P., "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[18] Dally, K., "Deep Reinforcement Learning for Flight Control," *AIAA 2022-2078*, 2021. https://doi.org/10.2514/6.2022-2078.

Part II

Preliminary Results

***Graded under AE4020**

Chapter 2

Reinforcement Learning

Reinforcement Learning (RL) is a Machine Learning (ML) field that focuses on learning through trial and error. It takes inspiration from the way in which humans learn, by repeatedly attempting a task, observing the efficacy of their actions by looking at their associated rewards and adapting until their decision making is good enough to consistently achieve the desired goal. RL is a field considered separate from the other two types of learning, namely supervised and unsupervised learning. RL approaches problems by dividing the task into an agent and environment paradigm where the agent navigates in a sequential manner. The agent then uses trial and error in order to procedurally select actions based on evaluative feedback, all without the need to rely on knowledge of what the correct action should be. The idea being that some actions in certain states are more desirable than others and should be more favoured if the state was to recur. This is fundamentally different from supervised and unsupervised learning, which focus on generalising from labelled training data and pattern/structure recognition of unlabelled data respectively. For a broad overview of how these sub-fields of ML can be distinguished, refer to Figure 2.1.

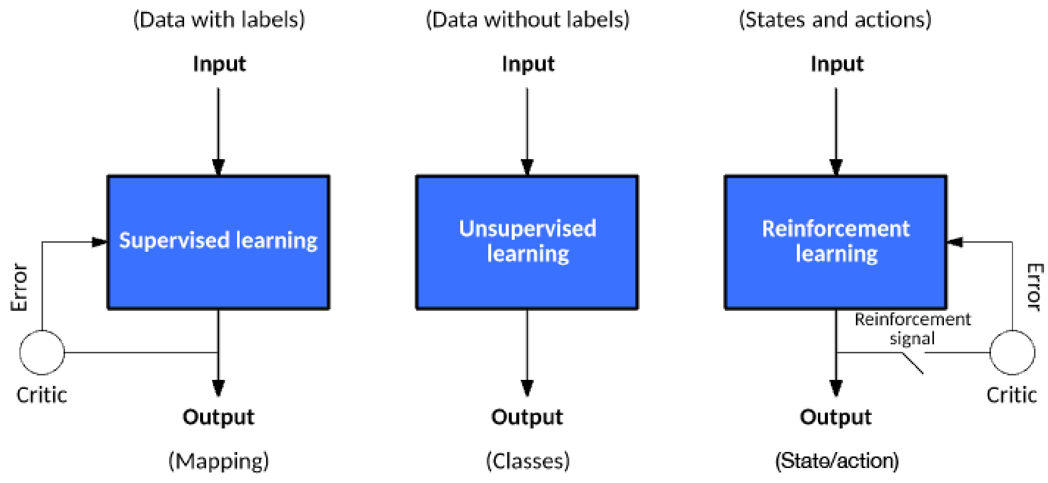


Figure 2.1: ML classification [6]

This chapter will introduce the concept of reinforcement learning and address RQ1.1 - RQ1.2. The chapter will begin by giving a brief intuition into RL in Section 2.1. This is followed by an explanation of the mathematical framework behind RL in Section 2.2. Section 2.3 will discuss some of the common approaches used in the field of RL. Methods aimed at dealing with continuous state-action spaces will be discussed in Section 2.4. Then Section 2.5 will explain some of the different words that are used to classify the RL taxonomy. Lastly, Section 2.6 will discuss what benchmarking is and list some relevant metrics when discussing RL to be used in real life applications.

2.1 The concept of reinforcement learning

In essence, RL can be seen as an exercise in which a learner, denoted agent, interacts with its surroundings, denoted environment. The agent acts based on some decision making, termed as policy, with the aim of achieving a certain goal or terminal condition. The agent is said to have states and these state-action transitions are associated with rewards. Consider Figure 2.2, here the agent would be the cleaning robot and the goal would be to either reach the power pack to recharge or reach the trash in order to throw away what it has collected.

The goal states are awarded with the rewards in green. In this case, the agent's state would be its horizontal position, its policy would be the decision making in charge and the environment would be the transition function describing all the valid moves to make together with the reward landscape.

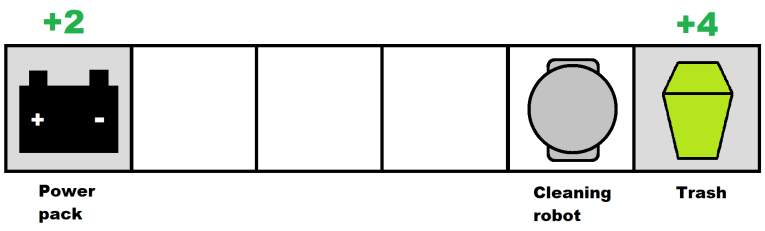


Figure 2.2: Cleaning robot example

Intuitively, the best policy based on how this problem is formulated, the optimal policy would be to choose right from the starting position in order to enter a terminal state and receive the most reward. The solution is, however, susceptible to changes in: the reward function formulation, the environments topology, the location & definition of the goal states and the policy the agent undertakes. Consider for example that the agent had no access to a model of the environment or that the topology was more complicated, it would then only be able to find a global optimal solution after fully exploring the environment. Currently there are also no penalties issued for wasted actions, meaning that in terms of optimality starting at the position illustrated an action sequence of $\rightarrow$ is equal to $\leftarrow, \rightarrow, \rightarrow$. However, one could easily imagine that if every non-terminal state was given a penalty of -1 the optimal solution would then solely be $\rightarrow$. Finally, consider the case where there are no terminal states and the goal of the robot is to keep cleaning continuously. In this case, one can imagine that the optimal solution would have to be a compromise between picking up trash, tossing it and recharging.

All of these different scenarios demonstrate how the optimal behaviour is sensitive to different aspects of how the task is formulated, where the more complex the system is the more non-intuitive the optimal solution will be. With this simple formulation and the mathematics detailed hereforth the power of RL as a tool will be displayed. Due to it's broadness and flexibility it sees uses in tasks ranging from home energy optimisation [7] to AlphaGo's mastery of Go [8]. Naturally, it also extends to flight control applications which is the focus of this study.

2.2 Mathematical framework for reinforcement learning

This section will go over the mathematical framework underlying RL, for this purpose the majority of the information and notation will be borrowed from the book Reinforcement Learning: An introduction by Richard S. Sutton and Andrew G. Barto [9]. The notion of Markov Decision Processs will be introduced in Subsection 2.2.1. This will be followed by the definition of returns in Subsection 2.2.2. Lastly, Subsection 2.2.3 will go over the formal definitions of policies and value functions.

2.2.1 Markov decision processes

A Markov Decision Process MDP is a probabilistic model that is used as the mathematical framework for reinforcement learning. They are meant to be a direct mathematical framing of the problem of learning from interactions to achieve a goal. For the sake of brevity and simplicity the MDPs will be discussed in the context of finite states, actions and rewards. Figure 2.3 illustrates the agent-environment interface for MDPs where the agent is the learner and the environment is what the learner interacts with. As the figure shows, the agent can navigate from its current state (denoted S_t) to another state S_{t+1} by taking an action A_t, upon which a reward R_t is issued.

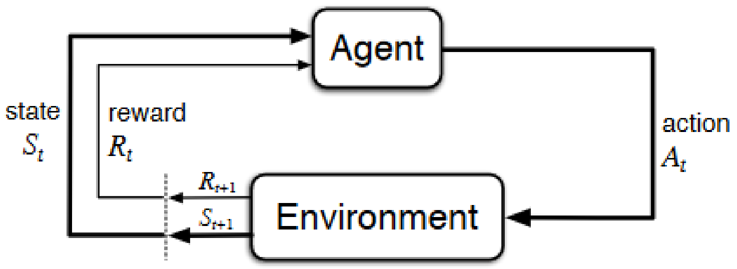


Figure 2.3: Agent environment interface [9]

The entire process, can be described as seen in Equation 2.1. Here, the discrete states $S_t \in S$ refer to the state at a discrete time $t \in \mathbb{N}$ which allows for action $A_t \in A(s)$. Note the set of actions is dependent on the state as some states might permit some actions while others do not. Once the action is taken a reward $R_{t+1} \in R \subset \mathbb{R}$ is received and the agent arrives at a new state S_{t+1} after which this process is repeated ad infinitum or until a terminal condition is met.

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots \tag{2.1}$$

In a finite MDP all the state, action and reward sets are of finite magnitude which allows for the random variables R_t and S_t to follow discrete probability distributions which only depend on the previous state and action. Equation 2.2 shows this phrased mathematically. Particular values for the random variables are written in the lower case (using s' as an example) $s' \in S$. The apostrophe in s' signifies a subsequent state after an action relative to a given state s. The function p encapsulates the dynamics of the MDP and is defined as $p : S \times R \times S \times A \to [0, 1]$ which reads as "the probability of moving to state S_{t+1} and receiving reward R_{t+1}, given the transition starts at a state S_t with the associated action A_t".

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{2.2}$$

It then follows that the probability of the combination of all subsequent states and rewards should add up to one (as they encompass all the possible combination of subsequent state reward pairs). This is shown mathematically in Equation 2.3.

$$\sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a) = 1, \text{ for all } s \in S, a \in A(s) \tag{2.3}$$

The state transition function can then be derived from the system dynamics as seen in Equation 2.4, where $f : S \times S \times A \to [0, 1]$ is a mapping from state-action pair to a following state. In short f is found by summing over all possible rewards in subsequent state s' in order to find the probability of landing on that state from the antecedent state-action pair.

$$f(s' \mid s, a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in R} p(s', r \mid s, a) \tag{2.4}$$

The expected rewards can also be computed from the system dynamics as seen in Equation 2.5. Here, the state-action pair reward is a two dimensional function $r : S \times A \to \mathbb{R}$ which is understood as the reward collected from all subsequent states given a state-action pair.

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r \mid s, a) \tag{2.5}$$

The expected reward per subsequent state can also be defined from p and is done so in Equation 2.6. This is useful for estimating the reward associated with specific state transitions, here $r_3 : S \times A \times S \to \mathbb{R}$ where r_3 is used to distinguish the fact that it considers the subsequent state as well.

$$r_3(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in R} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)} \tag{2.6}$$

2.2.2 Returns

The formal definition for episodic return and how it's related to reward is seen in Equation 2.7, where T is seen as the final time step so R_T is the last reward. Here, an episode is defined as the entire sequence of interactions between the agent and environment until a terminal state or final goal is achieved. Note that this definition assumes that there will be a final goal or terminal states for which the episode will end.

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{2.7}$$

Another commonly used metric for return is given in Equation 2.8. This equation depicts the discounted return, changing the notion of return into a converging geometric series that give more value to short term rewards and converge to a fixed value. Here $0 \leq \gamma < 1$ represents the discount factor for future returns, for which $|\gamma| < 1$ as the aim is to maximise the cumulative discounted reward over some horizon of upcoming states (with T being the last timestep). Using $\gamma = 1$ would reduce it to the episodic return (Equation 2.7).

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{2.8}$$

The definition of discounted return lends itself to recursion quite nicely. Equation 2.9 demonstrates how the return at time t can be recursively formulated in terms of future returns. The base case for the recursion being that at whatever time T the task is terminated, a return of $G_T = 0$ is given.

$$\begin{aligned} \textbf{Base case: } & G_T \doteq 0 \\ \textbf{Recursive case: } & G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \cdots\right) = R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.9}$$

Finally, for applications where the task is continuously on-going and there are no expected terminal conditions in the formulation of the problem, Equation 2.10 [10] is used. This shows the average return or return rate, which is the definition needed in continuous on going tasks like balancing an inverted pendulum in the absence of a terminal condition as the aim of the agent in that case would be to maximise its rate of reward indefinitely. It essentially calculates the return at time t by averaging the rewards over the next T timesteps by effectively using the future timestep T as a pseudo terminal state.

$$G_T \doteq \frac{1}{T-t-1}(R_{t+1} + R_{t+2} + \cdots + R_T) = \frac{1}{T-t-1}\sum_{k=0}^{T-t-1} R_{t+k+1} \tag{2.10}$$

2.2.3 Policies & value functions

The final piece to conclude the formulation of MDP is to discuss the definitions of policies and value functions. A policy is defined as seen in Equation 2.11. Here, it is said that if an agent is following a policy at time t then π will be a mapping $\pi : S \rightarrow A$ which describes the probabilities that the current policy chooses a possible action a given a state s. The optimal policy, often denoted as π_*, would be one that at every given state would choose the action that would result in the highest return.

$$\pi(a \mid s) = \Pr(A_t = a \mid S_t = s) \tag{2.11}$$

One policy that is often used in literature is the ϵ greedy policy which is defined as seen in Equation 2.12. Here, the policy acts greedily as in a way that follows the maximum estimated Q value (introduced further in this subsection but for now consider Q to be a proxy for return) with a $1-\epsilon$ probability and a random action with an ϵ probability. Often, epsilon is scaled with the number of episodes k such that over time it converges to a fully greedy policy $\lim_{k\to\infty} 1 - \frac{\epsilon}{k} \rightarrow 1$ as the number of episodes increases.

$$\pi(a|s) = \begin{cases} 1-\epsilon & a = \arg\max_{a'} Q(s,a'), \\ \epsilon & \text{random action} \end{cases} \tag{2.12}$$

The value function v with its associated policy π represents the expected return when starting in s and following π. This is formulated for MDPs as seen in Equation 2.13.

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in S \tag{2.13}$$

Similar to the discounted return function, the value function can also be defined recursively in terms of its successor states as seen in Equation 2.14. This equation is called the Bellman equation and it plays a central role for value based RL methods.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\right] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right], \quad \text{for all } s \in S \end{aligned} \tag{2.14}$$

The action-value function q with its associated policy π represents the expected return when starting in s, taking action a and following π. The mathematical formulation for MDPs is given in Equation 2.15.

$$q_\pi(s, a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right] \tag{2.15}$$

There is also an analogous Bellman equation for q, seen in Equation 2.16 [10]. Note again the double sum being abbreviated into one by collapsing the sum over the reward and successive state.

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a')\right], \quad \text{for all } s \in S,\ a \in A \end{aligned} \tag{2.16}$$

For value functions, optimality is defined as a value function that yields the optimal policy when the actions that transition to the most valuable states is followed. Mathematically, this is given in Equation 2.17.

$$v_*(s) \doteq \max_\pi v_\pi(s) \tag{2.17}$$

Similarly to v_*, the optimal q_* function is defined as seen in Equation 2.18.

$$q_*(s, a) \doteq \max_\pi q_\pi(s, a) \tag{2.18}$$

2.3 Common approaches

In this section, some approaches for dealing with RL problems are given. A brief overview of evolutionary methods is given in Subsection 2.3.1, policy optimisation & value-based methods in Subsection 2.3.2, dynamic programming in Subsection 2.3.3, Monte-Carlo methods in Subsection 2.3.4, temporal difference methods in Subsection 2.3.5 and finally a unified view of these last three methods will be given in Subsection 2.3.6.

2.3.1 Evolutionary methods

Evolutionary methods refers to the family of methods that seek to find an optimal policies through methods such as: genetic algorithms, genetic programming, simulated annealing and other optimisation strategies [9, p. 7]. The similarity between them is that they all start with a population of static policies from which best performing ones are selected iteratively (exploration is done through adding randomness akin to mutation), similar to the way evolution optimises for the organisms which are most fit. The main difference between evolutionary methods and classic RL methods is that the evolutionary methods do not learn or update their policies based on the interactions of the agent as it moves through it's environment, rather they simply proceed in the direction of the group of static policies that perform best overall. In Gavra [11] evolutionary methods are used in conjunction with deep RL to optimise non-linear policies for fault tolerant flight control applications.

2.3.2 Policy optimisation & value based methods

The family of methods that make use of the value function (either $v_\pi(s)$ or $q_\pi(s, a)$) in order to find an optimal policy are called value based methods. These look to refine their value functions such that the optimal policy is then found indirectly by choosing the action that maximises the value function. Gong et al. [12] demonstrates how Q-Learning, a value based method, was used in order to optimise the switching control strategy for a variable sweep morphing aircraft. In contrast, policy optimisation methods look to refine the policy directly in order to achieve optimal performance. This is usually done through a parameterised policy (for example a neural network) which can be updated such that the return is maximised. Ramezani Dooraki and Lee [13] used an enhanced version of Proximal Policy Optimisation (PPO) in order to fly a quadrotor drone. The remains of this section will all focus on value based methods.

2.3.3 Dynamic programming methods

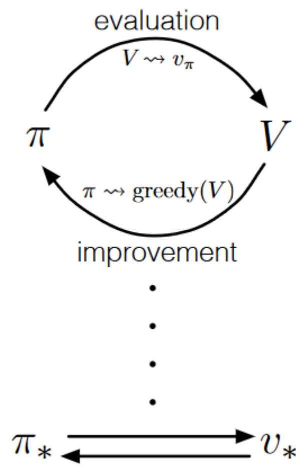
Dynamid Programming (DP) methods is a family of methods that given a true model of the environment, is able to arrive at optimal solutions. The overarching idea behind DP consists of two steps, policy evaluation and policy improvement. Policy evaluation consists of estimating the value function under a specific policy, this is done by converting the Bellman equation (Equation 2.14) into an iterative form. This is done by initialising the value function and using the full model to loop over all states to estimate their value (making use of the reward and transition functions). Policy evaluation terminates when either the value function stops changing or the change is considered smaller than some threshold Δ. Step 2 of Algorithm 1 shows the code of how a value function v under π is estimated, note how the value estimation $V(s)$ is just an iterative way of writing Equation 2.14.

Algorithm 1 Policy iteration algorithm for estimating $\pi \approx \pi_*$ [9]

```
1. Initialisation
$V(s) \in \mathbb{R}$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$; $V(\text{terminal}) \doteq 0$

2. Policy Evaluation
repeat
    $\Delta \leftarrow 0$
    loopeach $s \in S$
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{s',r} p(s',r \mid s,\pi(s))[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end loop
until $\Delta < \theta$ (a small positive number determining the accuracy of estima-
tion)

3. Policy Improvement
policy-stable $\leftarrow$ true
for each $s \in S$ do
    old-action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r \mid s,a)[r + \gamma V(s')]$
    if old-action $\neq \pi(s)$ then
        policy-stable $\leftarrow$ false
    end if
end for
if policy-stable then
    return $V \approx V_*$ and $\pi \approx \pi_*$; else go to step 2
end if
```


evaluation
$V \rightsquigarrow v_\pi$
π
V
$\pi \rightsquigarrow \text{greedy}(V)$
improvement
π_*
v_*
Figure 2.4: Generalised policy iteration [9]

In Algorithm 1 the policy improvement step occurs in step 3, here in essence it says that if choosing a greedy action for the value function at a given state leads to different outcome than the original policy, then the original policy has room for improvement if at the very least at that given state. If there is room for improvement for any state, the policy will be improved and evaluated once again. If there's no room for improvement, this means that the current policy is the optimal policy $\pi \approx \pi_*$ which directly implies $V \approx v_*$. In this example of policy iteration, evaluation and improvement are done sequentially but this is not strictly necessary. In general, algorithms may alternate between the two steps as they please and still converge to an optimal policy and value function as seen in Figure 2.4.

2.3.4 Monte-Carlo methods

Monte-Carlo (MC) methods are a family of methods where the agent does not have access to a complete model of its environment (as opposed to DP) and whose key characteristic is that they are updated episodically. This means MC methods must perform policy evaluation and control through exploratory sampling. The value function update rule for MC is given in Equation 2.19, where value function $V(S_t)$ is an estimate calculated for the value of state S_t under a policy π, τ is the learning rate and G_t denotes the episodic return (see Equation 2.1 for definition).

$$V(S_t) \leftarrow V(S_t) + \tau\,[G_t - V(S_t)] \tag{2.19}$$

The pseudo code seen in Algorithm 2 demonstrates how MC methods work. Note how the value of all the states are only updated once the episode has finished.

Algorithm 2 First-visit MC prediction, for estimating $V = v_\pi$ [9]

```
Input: a policy $\pi$ to be evaluated
Initialise:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in S$
  Returns($s$) $\leftarrow$ an empty list, for all $s \in S$
loop forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \cdots, S_{T-1}, A_{T=1}, R_T$
    $G \leftarrow 0$
    repeat for each step of episode, $t = T-1, T-2, \cdots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        if $S_t$ does not appear in $S_0, S_1, \cdots, S_{t-1}$ then
            Append $G$ to Return($S_t$)
            $V(S_t) \leftarrow$ average(Return($S_t$))
        end if
    until all steps have been covered
end loop
```

The associated control problem, or how the policy is improved, is analogous and attempts to estimate the associated state-action function $Q(s, a)$. The policy would then be updated to follow the actions most associated with a maximum reward at a given state (highest $Q(s, a)$ value), while also maintaining some chance for exploration. This allows for a more optimal global policy for which $\pi \rightarrow \pi_*$ as the number of episodes sampled grows.

2.3.5 Temporal difference methods

Temporal Difference (TD) methods are a group of methods where the agent does not have access to a complete model of the environment and operates very similarly to MC methods. The difference being that while MC updates only occur once the episode has terminated, TD updates occur using current estimates as the agent explores which is known as bootstrapping. This is exemplified in Equation 2.20, note how instead of the episodic return it instead uses the current value of the next state.

$$V(S_t) \leftarrow V(S_t) + \tau \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{2.20}$$

The pseudo code seen in Algorithm 3 illustrates an example of TD in action. Here, the name TD0 corresponds to the fact that it uses only the next immediate state in its update. In general, more than one future state might be used in the update rule and some known methods that utilise this are called n-step bootstrap or TD(n) methods. Here, the n represents the number of states that are considered in the update step for bootstrap methods.

Algorithm 3 Tabular TD(0) for estimating v_π [9]

```
Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\tau \in (0, 1]$
Initialise $V(s)$, for all $s \in S+$, arbitrarily except that V(terminal) = 0
loop for each episode:
    Initialise $S$
    repeat for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \tau[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until $S$ is terminal
end loop
```

Similarly to the MC control problem, policy improvement in TD is done by learning the $Q(s, a)$ function with some greedification while maintaining some compromise of exploration and exploitation to achieve a global optimal policy. Algorithm 5 shows how such an algorithm looks like.

2.3.6 Unified view of DP, MC & TD

The relationship between DP, MC and TD can be considered as shown in Figure 2.5. Here, the methods are distinguished by their width and depth, with full backups being associated with a wider search approach relative to those with sample backups and deep backups with a deeper search approach. Multi step methods like bootstrapping act as a bridge between pure MC and TD0 methods to balance out the need to arrive at the final outcome before updating while still propagating the future rewards across more than just the immediate next state.

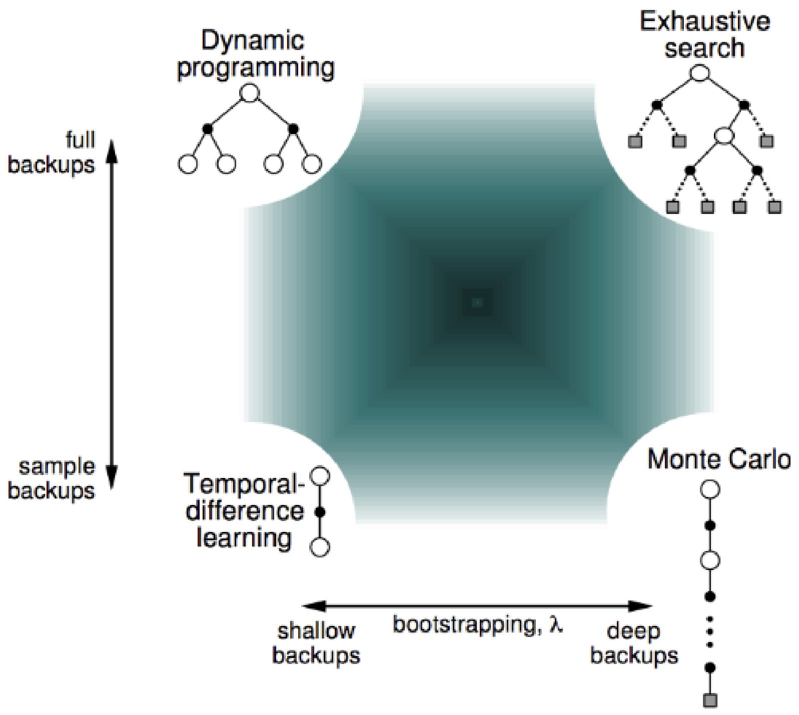


Figure 2.5: Unified view of DP, MC & TD [10]

2.4 Reinforcement learning for continuous state-action spaces

The discussion so far has dealt with discrete tabular problems that have finite sized state-action spaces. By this, it is meant that the set of states S and actions at every state $A(s)$ were of a countable nature and could reasonably be included in a look up table for a policy or value function. This does not hold for problems whose state-action space is either too large or continuous, as this would correspond to making an unreasonably large table or simply not being able to make one as the state-action space is not discrete finite. For these sort of problems, the discussion must be extended to include the following two here-forth discussed approaches: discretisation methods (Subsection 2.4.1) and approximation methods (Subsection 2.4.2).

2.4.1 Discretisation methods

The idea of discretisation methods is to lower the resolution of the state-action space by defining a grid for which finite discrete tabular solutions are still applicable. Note that it is not always the case that both the states and actions need to be discretised, there are scenarios where a state action space is only problematic along only one of them such as when there is a continuous state space but a small discrete number of actions (like balancing an inverted pendulum with bi-lateral fixed thrust inputs).

Ideally, a discretisation scheme should be constructed such that the optimal policy of the discretised case is also the optimal policy of the original problem. However, it can be the case that the discretisation can alter the formulation of the problems to a degree from which the original problem's optimal policy cannot be retrieved. To combat this, Sinclair et al. [14] designed an adaptive discretisation scheme for continuous state-action spaces that utilises tree based partition rules to go from an initial low resolution partition to a more refined ones as the agent explores the environment. The decision to split a partition is based on the certainty of the estimates of the value functions relative to the size of the partitions dimension. Figure 2.6 shows the result of the adaptive discretisation scheme, here the green regions in the diagonal are associated with a higher true value function value (as opposed to the estimates). It is clear from the figure that partition refinement only occurs in locations the agent has been to and where it's beneficial to do so, unexplored areas or where the certainty for the value function estimates is high remain the same size.

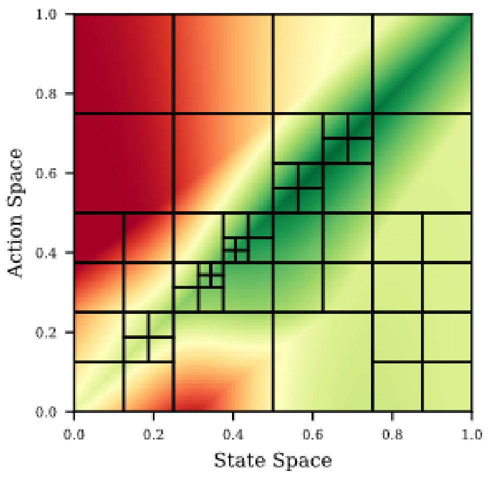


Figure 2.6: Resulting state action space of an adaptive discretisation scheme [14]

2.4.2 Approximation methods

The alternate approach often utilised is to use supervised learning techniques to approximate the policy or value as continuous functions of the state-action space. Parametric approximations are those which use a finite number of parameters to represent a function. These include parametric function approximators like neural networks, sum of basis functions and linear regression. Mnih et al. [15] showed how convolutional neural networks can be used to approximate a Q value function, dubbed Deep Q Network (DQN), to carry out a Q-Learning in high dimensional state-action spaces. In this case, the settings were Atari games and the inputs to the neural net were the raw screen pixels.

The other branch of approximation methods are the non-parametric ones. These don't have a finite number of parameters, but rather used the data gathered as the agent explores in order to form a model and as a result non-parametric methods scale with the amount of data that is given to them. Common non-parametric methods would be the likes of gaussian processes (GP), kernel methods and some Support Vector Machines (SVM). Engel et al. [16] demonstrated how GPs can be used to approximate a value function for a continuous two dimensional domain space for navigation. GPs have the advantage of being efficient in low data regimes [17] and offering a measure of uncertainty in their approximations which is of particular interest for tasks where safety is a high priority.

Each branch of the approximation methods has their advantages and disadvantages, however both are susceptible to overfitting on sample data which is not representative sample of the overall problem landscape. Parametric methods in general offer higher data efficiency as the lower complexity of a fixed number of parameters can help find simple useful models early on. On the other hand, non-parametric models offer more flexibility as they grow with the data but are very data hungry.

2.5 Reinforcement learning taxonomy

Within RL there are certain categories which are often mentioned in order to distinguish different types of approaches from each other. Figure 2.7 illustrates an overview of model-based and model-free approaches taxonomy, an explanation of these categories will be given in detail in Subsection 2.5.1 and Subsection 2.5.2 together with some pseudo code demonstrating how they work.

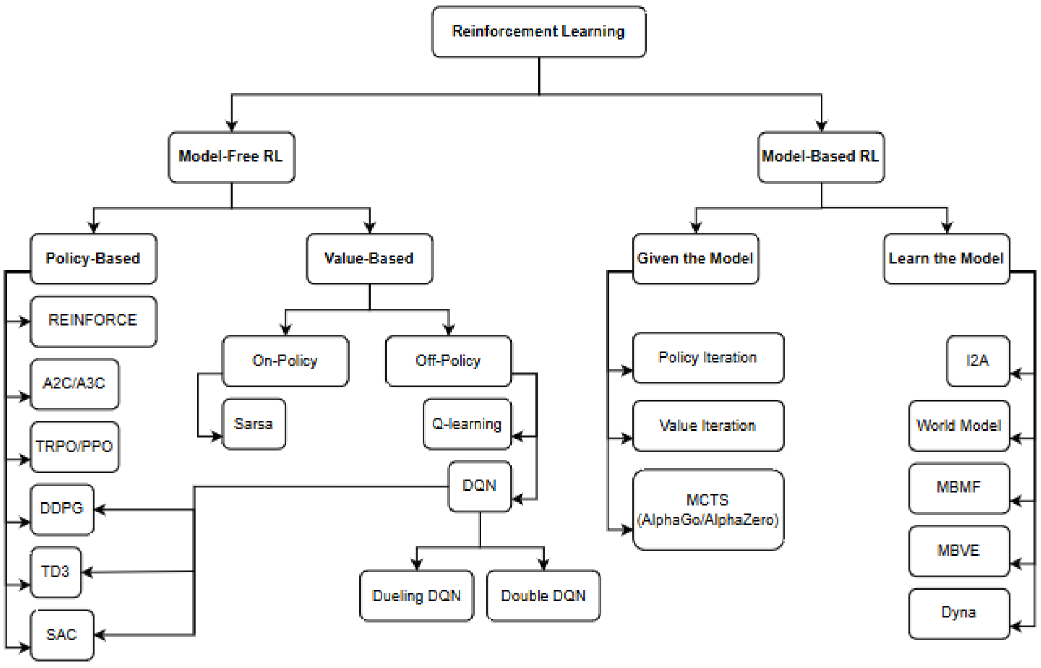


Figure 2.7: RL taxonomy depiction [4]

Apart from the model type classification there are other type of categories which are often discussed, such as: on-policy, off-policy, online, offline. These will be discussed in Subsection 2.5.3 and Subsection 2.5.4 respectively, once again together with some pseudo code implementations to highlight the differences between the categories.

2.5.1 Model based

Model Based Reinforcement Learning (MBRL) is, as the name suggests, a variant of RL where the agent has access to a partial/complete representation of its environment. The word access here is used in the context that the agent can either be given the model a priori or it can learn a model as it interacts with its environment. MBRL combines planning and reinforcement learning. Planning is defined as the ability to query the model at any preferred state-action pair [18, p. 8] in order to opt for better state-action routes across a horizon. MBRL has seen success in flight control applications, such as low level control of quadrotors [19] or autonomous Unmanned Aerial Vehicle (UAV) navigation [20].

The immediate advantage of MBRL is that due to the planning, the agent will be able to plan moves ahead and therefore require a lower amount of samples to achieve good performance. This is not without a price however, disadvantages to MBRL approaches do exist. Firstly, they need a model to begin with which involves either having the agent learn one or having a partial/complete model of the environment in order to give it to the agent. Secondly, in the case the model is learned, overfitting on mini batch samples can limit the solution space the agent will search and therefore result in sub-optimal solutions [21].

2.5.2 Model free

Model Free Reinforcement Learning (MFRL) is a variant of RL where the agent learns to make decisions solely based on observed experiences in the absence of accessing a model of the environment. MFRL is further categorised into two branches: methods that try to learn the value function in order to derive the optimal policy and methods which attempt to immediately learn the optimal policy [22]. MFRL has been successfully used for flight control applications like optimal control tracking [23].

The main advantage of MFRL is that it can be applied on any problem as no model is needed, allowing it to have a very plug and play nature to the approach. The main disadvantage of MFRL is that due to the emphasis on trial and error it tends to increase the number of samples the agent will have to go through in order to form an optimal value function or policy.

2.5.3 On-policy vs off-policy

The distinction between on-policy and off policy has to do with the way in which the algorithms update their policies. Those whose policy is developed using the data produced by the latest policy are dubbed on policy, while those that develop their policy using data from another policy are called off-policy. On-policy methods are more intuitive as the agent will update its policy based on the actions it actually takes during the training. SARSA [24], which stands for State Action Reward State Action, is an on-policy algorithm which will be used to illustrate the difference between the two approaches. Algorithm 4 illustrates SARSA in a discrete state-action space TD implementation, the on-policy label comes from the fact that the $Q(s, a)$ function update is done using the future discounted rewards of the $Q(s, a)$ function being learned.

Algorithm 4 SARSA (on-policy TD control) for estimating $Q \approx q_*$ [9]

```
Algorithm parameters: step size $\tau \in (0, 1]$, small $\varepsilon > 0$
Initialise Q(s,a), for all $s \in S^+, a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
loop for each episode:
    Initialise $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g $\varepsilon$-greedy)
    repeat for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \tau[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'$; $A \leftarrow A'$
    until $S$ is terminal
end loop
```

Off-policy methods may use foreign policies (often called behaviour policies) when updating the policy being learned, where the policy being learned or optimised is called the target policy. The advantage of this is that foreign policies can be used to stabilise, optimise or learn from when developing a new target policy. Q-Learning [25] is an example of an off policy algorithm, Algorithm 5 shows the pseudo code of a discrete state-action space TD implementation of Q-Learning. Here, the $Q(s, a)$ function update step is updated using the discounted future rewards calculated by a greedy policy even though the $Q(s, a)$ function being updated is not greedy in of itself.

Algorithm 5 Q-Learning (off-policy TD control) for estimating $\pi \approx \pi_*$ [9]

```
Algorithm parameters: step size $\tau \in (0, 1]$, small $\varepsilon > 0$
Initialise Q(s,a), for all $s \in S^+, a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
loop for each episode:
    Initialise $S$
    repeat for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g. $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \tau[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$
    until $S$ is terminal
end loop
```

2.5.4 Online vs offline

Online learning is defined as the family of machine learning methods where a “learner attempts to tackle some predictive (or any type of decision-making) task by learning from a sequence of data instances one by one at each time” [26]. In the case of RL, the learner would be an agent which usually starts with limited or no knowledge of its environment and is fed information about its state sequentially. Such methods are of importance to flight control applications due to the fact that real-time dynamic systems operate with a stream of sequential data that’s fed to the controller. Such systems are a great area of application for online learning, especially if there’s no prior training data to learn from. In application, due to the nature of how online learning works, safety and reliability risks are a real concern as the learning is directly done in a possibly dangerous, expensive and highly unstable system. Nonetheless, online RL has shown to be viable in, for example, a Cessna Citation II high fidelity model [27].

Offline learning, on the other hand, is a data driven approach that makes use of a previously collected static dataset of a problem in order to learn something that performs well when actually made to face the problem. In the context of RL, this would correspond to deriving a “sufficient understanding of the dynamical system underlying the MDP entirely from a fixed dataset, and then construct a policy that attains the largest possible return when it is actually used to interact with the MDP” [28]. Due to the nature of how offline RL is formulate, it’s important that the dataset be representative of the real underlying MDP for the learning to be meaningful.

As a result, one of the main disadvantages of offline RL are the need of having a large enough representative dataset. However, it does allow for learning in a safe environment which if translated well to the real problem will result in safe and reliable performance as has been shown in quadrotor's motion and path planning [29].

Figure 2.8 illustrates the differences between the two methods. It ties it back to what was previously discussed in the off-policy RL methods. The rightmost diagram depicts the usual offline RL setting, where a policy π_{off} is extracted from a static dataset D collected under an agent moving in an environment using policy π_β. The middle diagram depicts an off-policy RL algorithm which in this case makes use of a mix of static data (which will serves as π_{off}) and continuously collected experience data under a buffer D. It updates the policy $\pi_k \rightarrow \pi_{k+1}$ in an off-policy manner using π_{off} as the behaviour policy. The leftmost diagram shows online RL, where the agent must update on the fly using newly collected data from its latest policy and without any dataset or buffer. Note the online algorithm may still be off-policy, it will be online and off-policy so long as the behaviour policy it uses to train the target policy is not extracted from offline data.

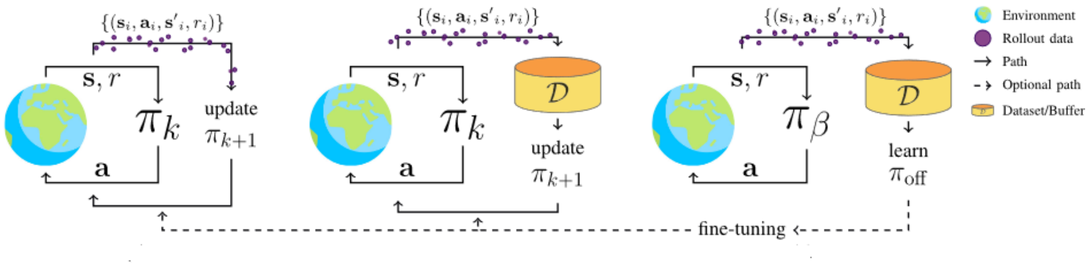
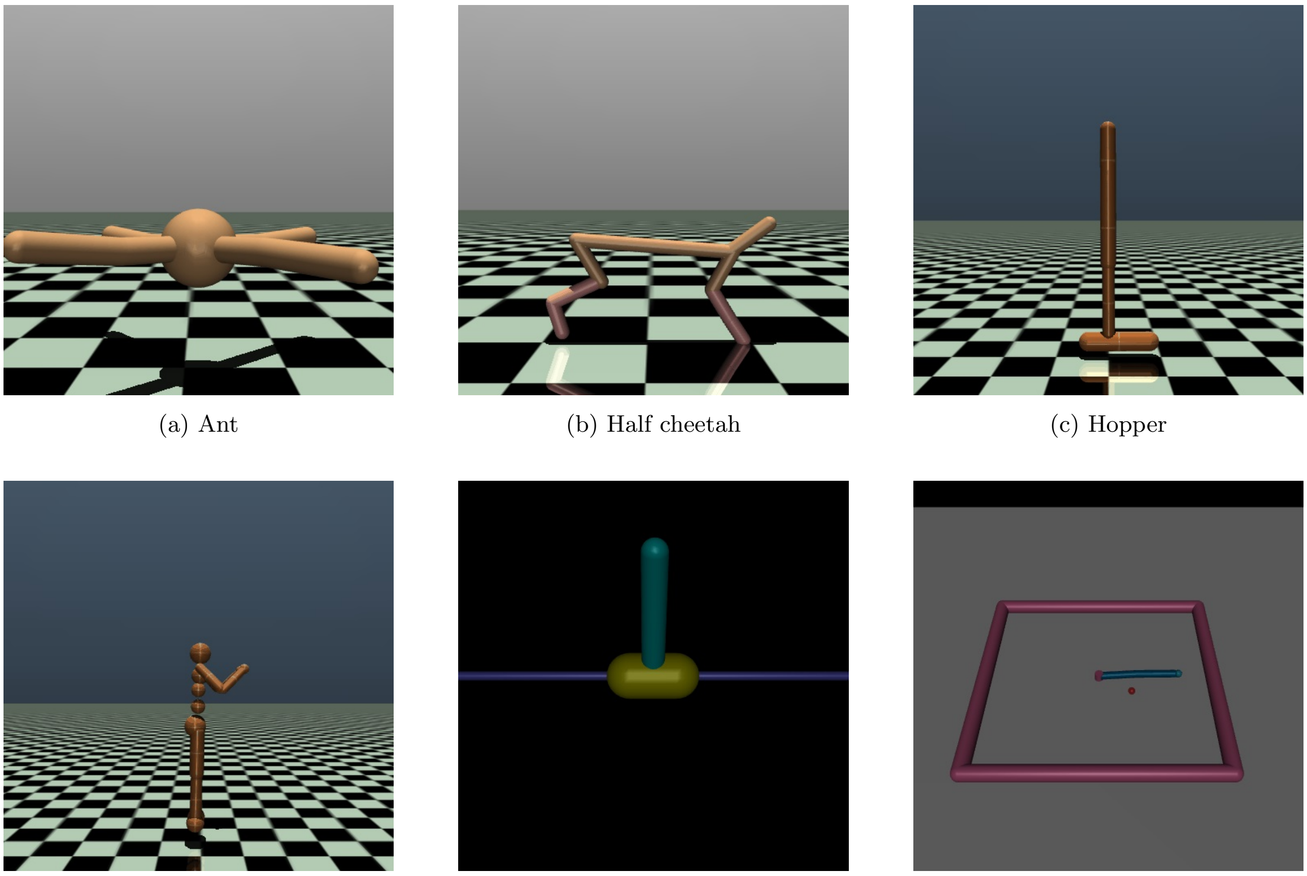


Figure 2.8: Online-Offline learning paradigm [30]

2.6 Benchmarking & common metrics

Benchmarking is the process of evaluating the performance of different models/algorithms by comparing relevant metrics in standardised tasks. Benchmarking serves as a way of measuring advancements, helps aid in the reproducibility of papers and serves as a yard stick along which different models can be compared to each other. One of the most commonly used benchmarks in the field of AI is the gymnasium python library, the gymnasium project provides an API for many common environment implementations for single agent reinforcement learning problems like cartpole, pendulum and atari games. Among many features it contains a collection of: environments, pre-built learning algorithms to train your agents with, logging options to track performance, ways to speed up or optimise the training and ways of live monitoring performance as the agent trains.

The MuJoCo environments collection are of particular interest for flight control problems involving continuous state-action spaces. This is because the collection of environment features many continuous state-action spaces, the idea being that using these environments would map better onto flight control problems and therefore be more beneficial benchmarks to use. The environment collection offers a range of complexity, from an inverted pendulum problem to teaching a bidepal agent how to walk as seen in Figure 2.9.


(a) Ant (b) Half cheetah (c) Hopper

(d) Humanoid (e) Inverted pendulum (f) Reacher

Figure 2.9: Selection of MuJoCo environments [31]

The metrics used for benchmarking performance naturally depend on the area of application. However, for algorithms that aimed to be applied in the real world (Dulac-Arnold et al. [32]) mentions the following nine metrics and equations which will be explained hereafter:

1. Average return
2. Warm start
3. Time to R_{min}
4. Safety violations
5. Worst case performance
6. Robust performance
7. CVaR return
8. Multi-objective
9. Explainability

Average return

Average return is a common metric used to compare algorithms, it averages the amount of return the algorithm receives across episodes. Note that this is different from the average return or return rate discussed in Equation 2.10, which had to do with a running average return on a single episode across a continuously on-going non-terminal task.

Warm start

The warm start measure seen in Equation 2.21 measures the cumulative return from the initial policy learned from offline external data. Here, D_{π_B} is the tuple of state-action reward sequences under the training data policy π_B. The idea being that its important for the initial policy extracted from the data to perform well enough to suggest that the offline training data will translate well into the actual problem.

$$J^{start} = R(Train(D_{\pi_B})) \tag{2.21}$$

Time to R_{min}

This is a metric for the sample efficiency of the algorithm, the measure in Equation 2.22 reads as the minimum number of data samples needed to train in order to achieve a minimum reward threshold.

$$J^{\text{eff.}} = \min |D_i| \text{ s.t. } R(\text{Train}(D_i)) > R_{\text{min}} \tag{2.22}$$

Safety violations

The safety of an RL algorithm in real world applications is a major factor, Equation 2.23 is a measure of the number of safety violations across K distinct safety constraints by a policy π. Each of the different constraint violations is stored along the entries of the K length $\boldsymbol{J}$ vector.

$$\boldsymbol{J}_{safety}(\pi) = \left(\sum_{i=1}^{T} c_j(s_i, a_i) \right)_{1 \le j \le K} \in \mathbb{R}^K \tag{2.23}$$

Worst case performance

The worst case performance value function is given by Equation 2.24, here $\mathscr{P}$ denotes a set of transition matrices from which p is selected such that it's as if at each step nature kept choosing the transition that leads to a minimal long term reward function. Using p, the agent learns a policy that maximises the performance return under this worst case scenario which is important for real applications due to the inherent dangers and costs of catastrophic worst case outcomes.

$$J(\pi) = \inf_{p \in \mathscr{P}} \mathbb{E}^p \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \mathscr{P}, \pi \right] \tag{2.24}$$

Robust performance

The robust performance metric is concerned with how the algorithm will perform under peturbed environments, the measure is given in Equation 2.25 where it reads as the average performance across K perturbed environments. The perturbed test environment p is sampled from the set of test sets P.

$$J_{\text{robust}}(\pi) = \frac{1}{K} \sum_{p \in P} \mathbb{E}^p \left[\sum_{i=1}^{T} \gamma^{i-1} r(s_i, a_i) \right] \tag{2.25}$$

CVaR return

The Conditional Value at Risk (CVaR) return measurement is shown in Equation 2.26, this is essentially a risk measure that reads as optimising the reward via a parameter Γ over $w\%$ worst outcomes. These outcomes correspond with the w quantile of R and are represented as $\nu_w(\Gamma)$ [33].

$$\Phi(\Gamma) = \mathbb{E}^{\Gamma} \left[R \mid R \le \nu_w(\Gamma) \right] \tag{2.26}$$

Multi-objective

The multi-objective measure is given by Equation 2.27. Note here $\boldsymbol{J}$ is bold to show it's a vector, where each of the K entries being a sub-reward metric (imagine an algorithm needs to have a compromise of performance, safety, time-to-target, etc).

$$\boldsymbol{J}_{\text{multi}}(\pi) = \left(\sum_{i=1}^{T_n} r_j(s_i, a_i) \right)_{1 \le j \le K} \in \mathbb{R}^K \tag{2.27}$$

Explainability

The explainability of an algorithm is important when human machine interaction is present in real world applications for the operator to be able to understand the decision making behind the algorithm, Dulac-Arnold et al. [32] suggest user surveys about how understandable the intent of the policy is by humans as a metric.

Chapter 3

Methodology Proposal

Now that the background information necessary to propose a method has been covered, the research objective and questions can be tackled. This chapter will narrow down on the methodology that will be used to achieve the research objective and cover: RQ1.3 & RQ3.1 - RQ3.4. As the research objective states the aim is to improve the sample efficiency of current flight control methods, this is of course quite broad and some choices have to be made on the scope to allow for a fruitful assignment. The following choices have been made which will subsequently be justified:

I. The task shall be performed on the Cessna Citation PH-LAB model

II. The agent shall incorporate both offline and online MFRL in the tasks

III. The approach for the task shall be based on REDQ (discussed in Section 3.2)

Firstly, the Cessna Citation PH-LAB model will serve as the environment for the RL agent, this is because it is a readily available TU Delft model that's been through verification & validation [34]. This means that the behaviour given by the model should correspond with high fidelity to the real PH-LAB Cessna Citation seen in Figure 3.1.

Figure 3.1: TU Delft's Cessna Citation - PH-LAB [35]

Secondly, including both online and offline MFRL aspects was chosen because sample efficiency is tackled differently for these two settings. In an online setting, sample efficiency is mostly defined in an implicit and binary way such that if an algorithm is able to learn a task online then it must also be sample efficient. However, it can also be comparatively defined by comparing how fast a policy is learned in real time relative to another online algorithm. In an offline setting, sample efficiency is measured by how much training an algorithm needs to achieve a certain performance which is a lot more explicit and comparative than the former. The idea of choosing both is that if the algorithm performs well enough offline (in terms of sample efficiency) then it could be that it is efficient enough to learn online. Lastly, the approach will be based off of REDQ due to it being an online MFRL algorithm and its great performance in terms of sample efficiency for several MuJoCo continuous state-action environments [5]. Per the date of this document this would be the first time REDQ has been applied to flight control applications as far as the author is aware.

The coming sections will piece together how both offline and online MFRL can be used in the PH-LAB model. Section 3.1 will discuss how flight control can be formulated for RL, Section 3.2 will detail how REDQ works and Section 3.3 will discuss which RQs have been addressed and which will need to be addressed in the thesis assignment.

3.1 Flight control in the context of reinforcement learning

In the context of flight control, RL problems are formulated in such a way that the agent represents the controller for the task being learned, the action is the control input given by the agent and the environment is the system dynamics that act as both the state transition function and reward function (via the reference error). Figure 3.2 illustrates a simple case of how this would look in a control loop for a pitch tracking control task. Here the pitch error e would be the state of the agent (RL controller), the feedback system dynamics (servo, aircraft and sensor) the environment and the control input c is the action taken by the agent.

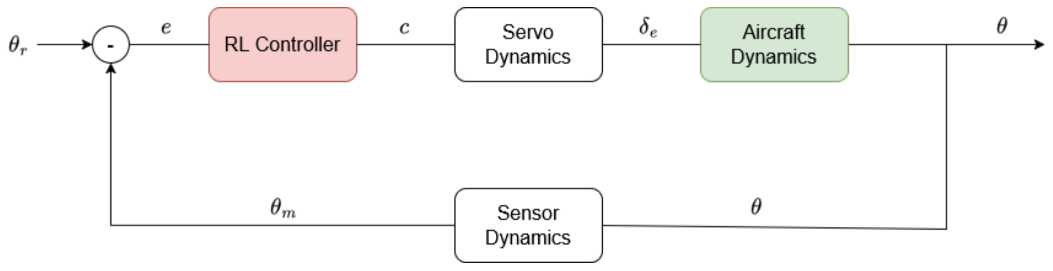


Figure 3.2: Pitch tracking RL controller block diagram

Using the PH-LAB, an identified model for TU Delft's Cessna Citation [34], control tasks can be formulated for many different objectives. The full list of states and actuators that are present in the model are shown in Equation 3.1.

$$\begin{aligned} x &= \begin{bmatrix} p & q & r & V_{TAS} & \alpha & \beta & \phi & \theta & \psi & h_e & x_e & y_e \end{bmatrix} \\ u &= \begin{bmatrix} \delta_e & \delta_a & \delta_r & T_{N_1} & T_{N_2} \end{bmatrix} \end{aligned} \tag{3.1}$$

This section will discuss two relevant examples of how the PH-LAB model has been used together with RL in order to solve flight control problems.

Online fault tolerant pitch control

For their thesis Chan [36] designed an online pitch controller for the PH-LAB using an Incremental Dual Heuristic Programming (IDHP) agent that was able to perform when previously unseen (to the agent during training) faults were introduced. Their approach involved using the IDHP agent to learn a model online of the dynamics of the associated MDP problem via a recursive least squares approach, this model is then used to improve the actions of the actor as well improve the value estimation of the critic to maximise the cumulative reward.

For their MDP problem formulation Chan [36] used (with the notation previously introduced in Equation 3.1) Equation 3.2. Here θ_e is the pitch error with respect to the reference.

$$\begin{aligned} s &= [\alpha \quad \theta \quad q \quad \theta_e]^T \\ a &= [\delta_e] \\ r &= -(\theta - \theta_r)^2 \end{aligned} \tag{3.2}$$

For their experiment, they designed a warmup phase during which the agent would learn and a manoeuvring phase where faults to the model would be introduced to determine if the aircraft was able to track in the presence of faults unforeseen to the agent. The thesis demonstrated that the IDHP agent was able to improve the fault tolerance performance.

Robust offline trained attitude and altitude control

For their thesis Dally [37] developed a cascaded controller that trained two Soft Actor Critic (SAC) (see Subsection 3.2.3 for more detail on actor critic architectures) agents offline for attitude and altitude tracking. One agent was in charge of controlling the attitude of the PH-LAB and the other agent was in charge of feeding a pitch reference to the attitude controller to mantain the altitude. The attitude controller was fully trained first and then the altitude controller was trained. The fault tolerance performance of the agents was then evaluated in scenarios the likes of a jammed rudder, gravitational center shift and icing on the wings. The agents were shown to be able to perform the attitude and altitude tracking despite the faults.

For their MDP problem formulation Dally [37] used Equation 3.3. Where $\mathbf{e}$ is the attitude error vector with respect to the reference, $\mathbf{c}$ is the associated cost vector and $\mathbf{u}$ is the present control input which was necessary as they controlled the actuator increment instead of the actuator directly (to smoothen out the actuator behavior).

$$\begin{aligned} s_{att} &= [(\mathbf{c}^{att} \bullet \mathbf{e}^{att}), \mathbf{u}^T, p, q, r]^T \\ a_{att} &= [\delta_e, \delta_a, \delta_r]^T \\ r_{att} &= -\frac{1}{3}||\text{clip}[\mathbf{c}^{att} \bullet \mathbf{e}^{att}, \textbf{-1}, \mathbf{0}]|| \\ r_{alt} &= -||\text{clip}[\mathbf{c}^{alt} \bullet \mathbf{e}^{alt}, \textbf{-1}, \mathbf{0}]|| \end{aligned} \tag{3.3}$$

3.2 REDQ

This section will introduce several aspects of REDQ in order to give some context for how REDQ works. REDQ stands for Randomized Ensenmble Double Q-Learning, Subsection 3.2.1 will discuss the ensemble aspect, Subsection 3.2.2 will go over the double Q-Learning aspect, Subsection 3.2.3 will discuss the actor critic architectures that REDQ can be layered on and Subsection 3.2.4 will tie everything together in an overview of how the actual algorithm works.

3.2.1 Ensemble learning

Ensemble learning approaches are RL approaches that use multiple learners that are trained concurrently. Figure 3.3 shows what this means in practice. In this case, an SAC solution is turned into an ensemble approach by having multiple actors and critics being trained simultaneously. Note here the use of a replay buffer. This is something that is commonly used when dealing with neural network function approximations to prevent the networks from forgetting past experiences. By creating a replay buffer from sampled past experiences and using it to continually train the network, catastrophic forgetting is avoided and learning is stabilised.

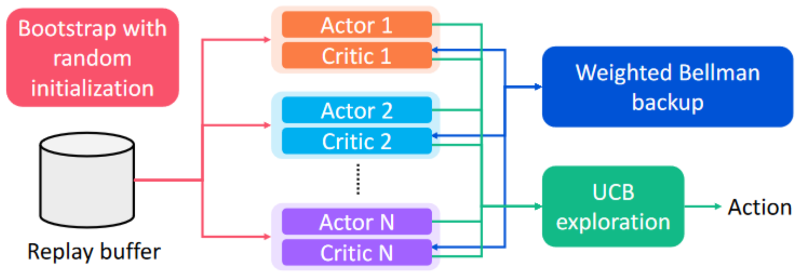


Figure 3.3: Ensemble soft actor critic visualisation [38]

This ensemble of actors and critics is then used to achieve more robust estimates for the value function and policy. In this case the estimates for the Q functions (critics) are used to generate a robust backup of Q and the multiple policies (actors) are used to choose which action maximises the Upper Confidence Bound (UCB) as defined in Equation 3.4 [38] which once again makes use of all the critics. Here ζ is a hyperparameter.

$$a_t = \max_a\{Q_{\text{mean}}(s_t, a) + \zeta Q_{\text{std}}(s_t, a)\} \tag{3.4}$$

3.2.2 Double Q-Learning

In Subsection 2.5.3 the off-policy algorithm Q-Learning was shown for a tabular case. This was an algorithm that tried to learn the state-action value function estimate Q by exploring the environment and constantly updating its estimate based on interactions. DQN is an extension of Q-Learning for problems that are non-tabular i.e problems in which the value functions or policies are impossible or impractical to arrange into a table. Mnih et al. [15] showed this to be successful in Atari games using the screen pixels as inputs (the high dimension of the states making a tabular solution impractical). DQN is a functional approximation approach (q being the function it tries to approximate) that uses a convolutional neural network to estimate $q \approx Q(s, a; \lambda_t)$ by comparing its estimates to a target (the value it actually received) once an action has been made. A paper by van Hasselt et al. [39] expanded on learning Q via function approximation by introducing the concept of Double Q-Learning (DQL). Equation 3.5 - Equation 3.8 explain the differences of the functional approximations between DQN and DQL.

$$\lambda_{t+1} = \lambda_t + \tau \left(y_t^Q - Q(S_t, A_t; \lambda_t) \right) \nabla_{\lambda_t} Q(S_t, A_t; \lambda_t) \tag{3.5}$$

$$y_t^Q \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \lambda_t) \tag{3.6}$$

These first two equations illustrate how a neural network's parameters are updated in a TD manner. Equation 3.5 shows how the networks parameters λ_{t+1} are updated by using the previous parameters λ_t, a target y_t^Q and the gradient of the network ∇_{λ_t} (computed via backpropagation). Equation 3.6 shows the usual definition of the target which is akin to the Bellman equation for Q.

$$y_t^{\text{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \lambda^-) \tag{3.7}$$

Equation 3.7 shows the definition of the target that was used by Mnih et al. [15], where the difference is that it uses a Q function approximation based on λ^- when computing the target. These parameters λ^- belong to a network that is a delayed version of the online network which is updated less frequently.

$$y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q \left(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \lambda_t); \lambda_t' \right) \tag{3.8}$$

The way in which DQL defines it's target is seen in Equation 3.8. It splits up the maximisation into two different networks: the online Q network that greedily chooses the future action and the delayed network which evaluates it. Splitting the maximisation by looking at two different estimates has shown to prevent overestimation and has been shown to perform better as seen in Figure 3.4.

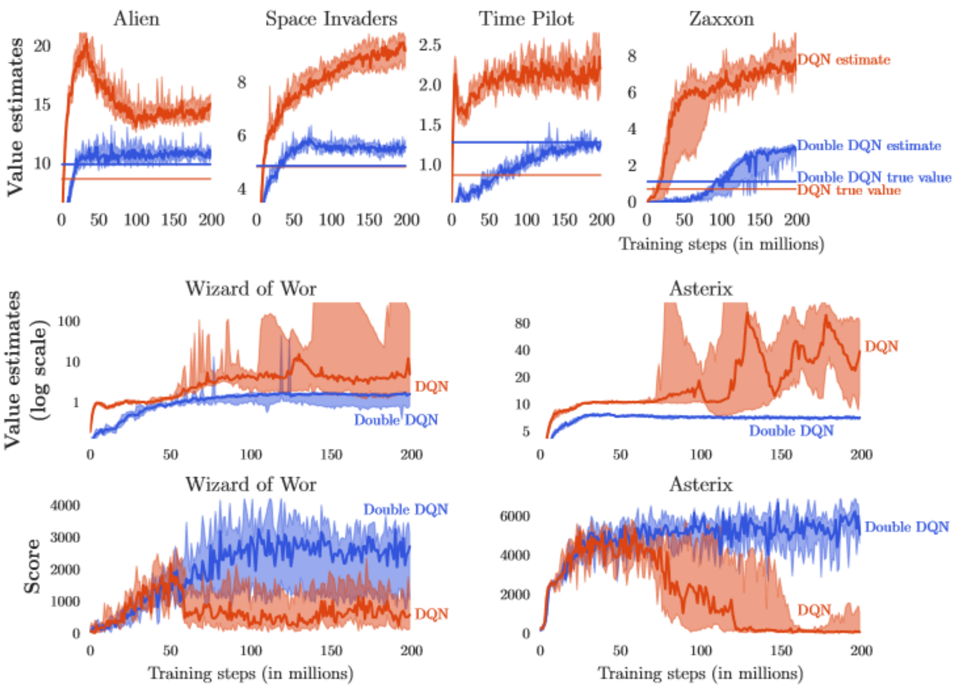


Figure 3.4: Deep Q Network vs Double Q-Learning comparison on Atari games [39]

3.2.3 Actor critic architectures

Actor critic architectures are a hybrid between policy optimisation methods and value based methods (see Subsection 2.3.2). They delegate the learning into two coupled roles: the actor which is in charge of the policy and the critic which is in charge of the value function. These two are usually coupled as the actor's policy usually takes into account the evaluation of the critic. In continuous state-action spaces the policies and value functions tend to be approximated by separate neural network which are updated based on some loss objective. A commonly used actor critic architecture is Deep Deterministic Policy Gradient (DDPG), shown in Algorithm 6. As mentioned, it makes use of critic and actor networks which are trained based on the squared loss of the target y_i and the associated predicted critic estimate $Q_\lambda(s, a)$. Note the use of target networks that act as softly updated backups via a hyperparameter ρ, similar to the previously discussed delayed networks.

Algorithm 6 DDPG Algorithm adapted from Lillicrap et al. [40]

```
1: Randomly initialize critic $Q$ and actor $A$ with weights $\lambda$ and $\Omega$ respectively
2: Initialize target network $Q'$ and $A'$ with weights $\lambda_{targ} \leftarrow \lambda$, $\Omega_{targ} \leftarrow \Omega$
3: Initialize buffer D
4: for episode = 1 till final episode do
5:     Initialize a random process for action exploration
6:     Receive initial observation state $s_1$
7:     for t = 1 till final timestep do
8:         Select action $a_t \sim \pi_\Omega(s_t) + \epsilon_{noise}$ according to the current policy and noise process
9:         Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
10:        Store transition $(s_t, a_t, r_t, s_{t+1})$ in D
11:        Sample a random minibatch $B$ transitions $(s_i, a_i, r_i, s_{i+1})$ from D
12:        Set $y_i = r_i + \gamma Q'_{\lambda_{\text{targ}}}(s', a')$
13:        Update critic by minimizing the loss: $L = \frac{1}{B}\sum_i (y_i - Q_\lambda(s,a))^2$
14:        Update the actor policy using the sampled policy gradient:
               $$\nabla_\Omega \approx \frac{1}{B}\sum_B \nabla_a Q_\lambda(s,a)|_{s=s_i, a=\pi_\Omega(s_i)} \nabla_\Omega \pi_\Omega(s)|_{s=s_i}$$
15:        Update the target networks:
               $$\lambda_{\text{targ}} \leftarrow \rho\lambda + (1-\rho)\lambda_{\text{targ}}$$
               $$\Omega_{targ} \leftarrow \rho\Omega + (1-\rho)\Omega_{targ}$$
16:    end for
17: end for
```

Two commonly used actor critic architectures are Soft Actor Critic (SAC) [41] and Twin Delayed Deterministic Policy Gradient (TD3) [42] which are both very similar to DDPG. SAC uses a stochastic policy that outputs an estimated mean and standard deviation which are used to calculate the entropy, this is then used for a entropy regularisation term in the target (y_i) definition. TD3 introduced the idea of using a minimisation over two critic networks to prevent overestimating the target y and delaying the actor network updates such that its updated less frequently than the critic [43] which has shown to boost performance.

3.2.4 REDQ overview

With the context set in place, REDQ can now be discussed. REDQ refers to the idea of using off-policy actor critic architectures together with ensemble learning and using the takeaways from DQL. Algorithm 7 shows a pseudo code implementation of REDQ SAC. Here, there are three main highlighted hyperparameters which must be defined: N, U and M. N refers to the number of critics (or Q functions), U refers to the ratio of Update To Data (UTD) and M refers to the subset size used for in-target minimization [5].

Algorithm 7 REDQ SAC Algorithm adapted from Chen et al. [5]

```
1: Initialise policy parameters $\Omega$, $N$ Q-function parameters $\lambda_i$, $i = 1, \ldots, N$, empty replay buffer $\mathtt{D}$.
2: Set target parameters $\lambda_{\text{targ},i} \leftarrow \lambda_i$, for $i = 1, 2, \ldots, N$
3: loop
4:     Take one action $a_t \sim \pi_\Omega(\cdot|s_t)$. Observe reward $r_t$, new state $s_{t+1}$
5:     Add data to buffer: $\mathtt{D} \leftarrow \mathtt{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
6:     for $U$ updates do do
7:         Sample a mini-batch $B = \{(s, a, r, s')\}$ from $\mathtt{D}$
8:         Sample a set $\mathtt{M}$ of $M$ distinct indices from $\{1, 2, \ldots, N\}$
9:         Compute the Q target $y$ (same for all of the $N$ Q-functions):
               $y = r + \gamma \left( \min_{i \in \mathtt{M}} Q_{\lambda_{\text{targ},i}}(s', a') - \kappa \log \pi_\Omega(a'|s') \right), \quad a' \sim \pi_\Omega(\cdot|s')$
10:        for $i = 1, \ldots, N$ do do
11:            Update $\lambda_i$ with gradient descent using
                   $\nabla_\lambda \frac{1}{|B|} \sum_{(s,a,r,s') \in B} (Q_{\lambda_i}(s, a) - y)^2$
12:        end for
13:        Update target networks with $\lambda_{\text{targ},i} \leftarrow \rho\lambda_i + (1 - \rho)\lambda_{\text{targ},i}$
14:    end for
15:    Update policy parameters $\Omega$ with gradient ascent using
           $\nabla_\Omega \frac{1}{|B|} \sum_{s \in B} \left( \frac{1}{N} \sum_{i=1}^{N} Q_{\lambda_i}(s, \tilde{a}(s)) - \kappa \log \pi_\Omega(\tilde{a}(s)|s) \right), \quad \tilde{a}(s) \sim \pi_\Omega(\cdot|s)$
16: end loop
```

Traces from the previous discussions are present in the code, such as the use of a replay buffer, entropy regularisation (the $-\kappa \log \pi_\Omega(\tilde{a}'|s')$ term) scaled by a temperature coefficient hyperparameter κ, the ensemble of N critics, preventing overestimates by not using the online critic function when calculating the target and the use of a slowly updated target network. Some new ideas are also present such as updating more than once per step via the hyperparameter U and some randomness is introduced by the sampling of M. The critics are updated with gradient descent and the actors using gradient ascent, which are ways to update the policy parameters so as to minimise the loss function or maximise the expected return (depending on the formulation of the optimisation problem) [44].

The main characteristic that makes REDQ interesting is that it has demonstrated high sample efficiency in multiple MuJoCo environments in a truly model free fashion, meaning that it not only lacks access to an a priori model of the environment but also does not even try to learn a model of the environment as it interacts with it. Previously, high sample efficiency approaches were mostly model based such as Model Based Policy Optimisation (MBPO) which learns a model as the agent interacts with the environment in order to increase its sample efficiency by using simulations of interactions with the learned model to learn more efficiently [5]. Figure 3.5 shows the performance of REDQ compared to MBPO with SAC being used as a measure of the average approach that does not focus on sample efficiency, in this figure both REDQ and MBPO have a UTD ratio of 20.

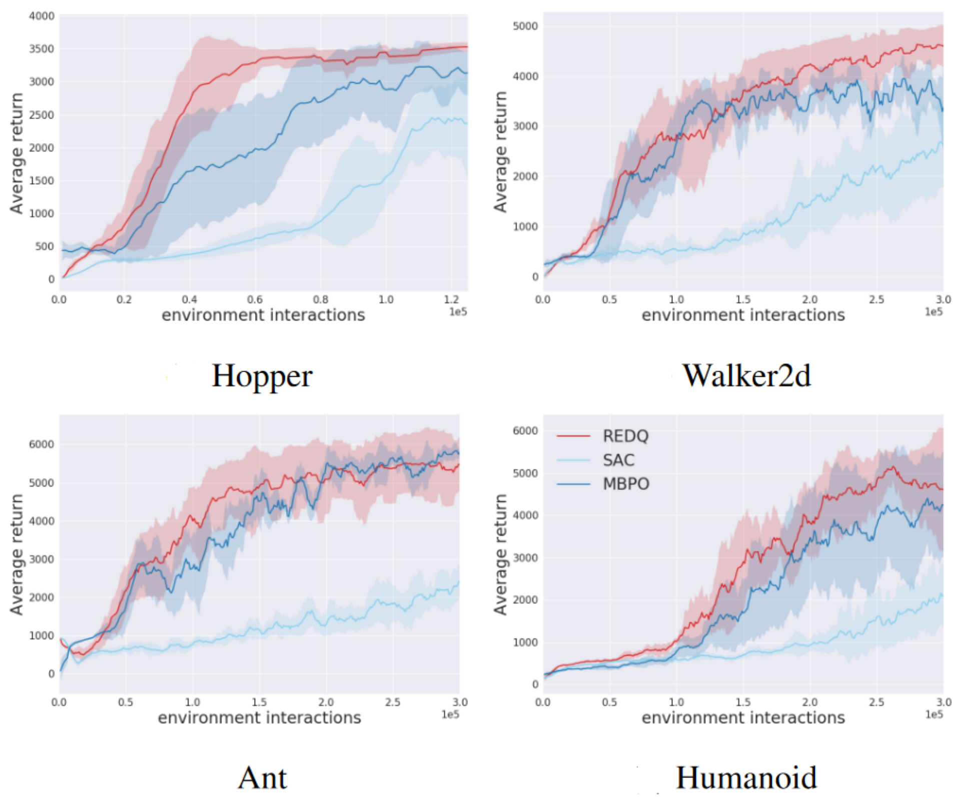


Figure 3.5: REDQ, MBPO and SAC comparisons in MuJoCo environments [5]

3.3 Addressing research questions

So far in this literature study: RQ1.1-RQ1.2 have been addressed in Chapter 2 and RQ1.3, RQ3.1 have been answered in this chapter. As RQ2.1-RQ2.3 will be answered in Chapter 4, the only remaining questions left to fully answer are RQ3.2-RQ3.4 as they require performing the full exercise. However, the list of steps outlined in the Chapter 9 serve as a plan of how these remaining questions will be tackled.

Chapter 4

Preliminary Analysis

This chapter will discuss the preliminary analysis that was performed as a way to test if the REDQ algorithm discussed in Chapter 3 will increase the sample efficiency in two different new environments not mentioned in the REDQ paper [5] and for the author to familiarise themselves with the coding, verification, hyperparameter tuning and agent evaluation skills for RL tasks in order to ease the transition when moving to the thesis assignment on the PH-LAB model. Section 4.1 goes over the environment choices and the rationale behind them. Section 4.2 discusses the procedure followed during the preliminary analysis. Section 4.3 details how the hyperparameter choices for the agents were chosen. Section 8.1 discusses the verification steps that were taken to ensure the agents were indeed learning as intended.

4.1 Environment choices

For the preliminary analysis, two environments were used to test the algorithms: Gymnasium's "Pendulum-v1" (hereafter referred to as hinged pendulum) and MuJoCo's "InvertedPendulum-v4" (hereafter referred to as inverted pendulum). This section will detail how these environments are defined and give motive for why these environments were chosen, Subsection 4.1.1 will discuss the hinged pendulum and Subsection 4.1.2 the inverted pendulum.

4.1.1 Hinged pendulum

The hinged pendulum environment consists of a freely rotating pendulum hinged at one end as seen in Figure 4.1 with the aim of keeping the pendulum upright. It's a continuous state-action task whose dynamics are easy enough for rapid learning by an agent. The fact that it is easily learnable makes it a perfect stepping stone onto more challenging tasks, as this simple environment allows for quick feedback for code implementation verification, algorithm convergence and generating data.

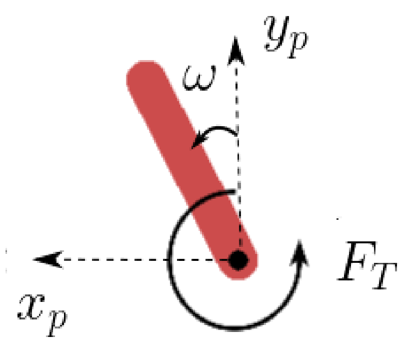


Figure 4.1: Diagram for hinged pendulum [45], signs are positive in the directions drawn

The state space consist of a three bounded dimensions, $-1 \leq x_p \leq 1$ [m] denoting the free end of the pendulums horizontal position, $-1 \leq y_p \leq 1$ [m] being the free end's vertical position and $-8 \leq \omega \leq 8$ [rad/s] representing the angular velocity. The action space is bounded, one dimensional and represents the torque force applied at the hinge denoted by F_T $[N \cdot m]$. This is summarised in Equation 4.1.

$$s = [x_p \quad y_p \quad \omega]^T, \quad a = [F_T], \quad -8 \leq F_T \leq 8 \tag{4.1}$$

The reward function is defined as seen in Equation 4.2, where β_p is the angle measured in radians between $[-\pi, \pi]$ with the upright position corresponding to zero.

$$r = -(\beta_p^2 + 0.1\dot{\beta_p}^2 + 0.001F_T^2) \tag{4.2}$$

An episode in the environment starts with a randomised selection of states and it ends when 200 timesteps have elapsed.

4.1.2 Inverted pendulum

The inverted pendulum environments consists of a task of balancing a pendulum upright that is mounted on a cart attached to a slider as seen in Figure 4.2. It's a continuous on-going task that is easy to interface with and is a level above complexity-wise relative to the hinged pendulum due to the existence of dynamic coupling between states.

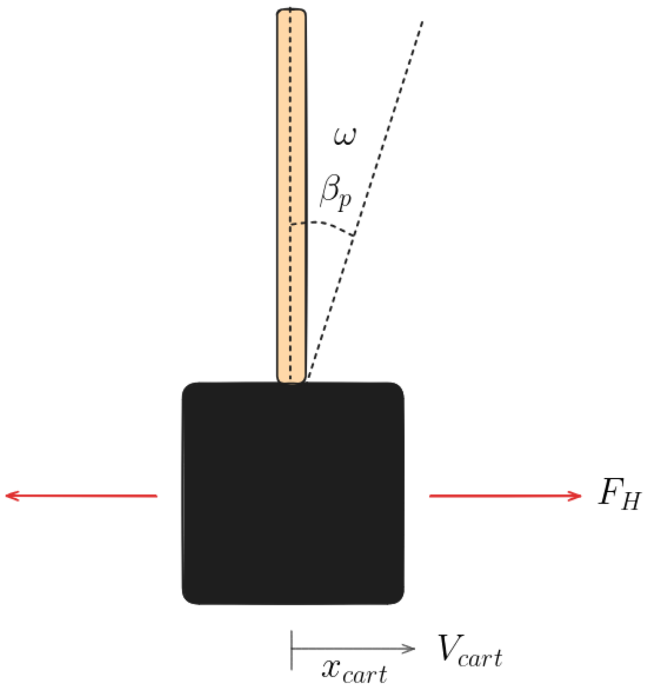


Figure 4.2: Diagram for inverted pendulum, signs are positive for the drawn directions

The environment is defined using four states with one possible action as seen in Equation 4.3. The states consist of the horizontal position of the cart x_{cart} [m] with right being positive, the angle of the pole with respect to the vertical β_p [rad] which is positive as seen in the diagram, the velocity of the cart V_{cart} [m/s] positive to the right and the angular velocity of the pole ω [rad/s] as depicted in the diagram. All of the states are unbounded meaning any of them can take values in $[-\infty, \infty]$. The action is defined as a horizontal force F_H [N] introduced at the middle of the cart positive to the right and is bounded with a max absolute value of three. This is summarised in Equation 4.3.

$$s = [x_{cart} \quad \beta_p \quad V_{cart} \quad \omega]^T, \quad a = [F_H], \quad -3 \leq F_H \leq 3 \tag{4.3}$$

The reward function is described by Equation 4.4.

$$r = \begin{cases} 1, & |\beta_p| < 0.2 \text{ rad} \\ 0, & \text{else} \end{cases} \tag{4.4}$$

An episode in this environment starts by sampling from some noise for the initial positions and velocities such that it doesn't always start from the same initial conditions. The episode ends under two conditions. If the magnitude of the angle of the pole is greater than 0.2 radians the episode is terminated, and if the reward reaches a value of 1000 the episode is truncated as the agent has learned to keep it upright.

4.2 Preliminary analysis procedure

This section will give an overview of the procedure followed during the preliminary analysis along with the main results. For a more intricate breakdown into the hyperparameter optimisation and verification see Section 4.3-Section 4.4. The preliminary analysis procedure for both environments consisted of coding the algorithms, verifying the code, doing a hyperparameter optimisation and then comparing the sample efficiency performance of the standard SAC algorithm with the REDQ version of it. For the coding, SAC and Algorithm 7 were implemented using PyTorch. SAC was chosen as the comparison baseline as it already had good performance and reference hyperparameters were readily available in the REDQ paper [5]. The neural network architecture for the actors and the critics are shown in Figure 4.3 and Figure 4.4 respectively. They represent the neural networks used to solve the hinged pendulum task (as seen by the number of states and notation), the neural network architecture for the inverted pendulum is analogous with an extra state on it's input layer. Hidden layers contain 256 units each. Note that as in SAC uses a stochastic policy the actor outputs two values, one for the action mean F_{T_μ} and the other for the log standard deviation F_{T_σ} which are used to create a normal distribution from which the actual policy action F_T is sampled from.

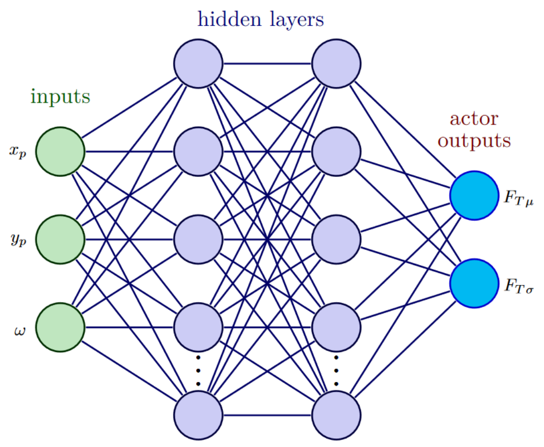


Figure 4.3: Actor network architecture for hinged pendulum environment

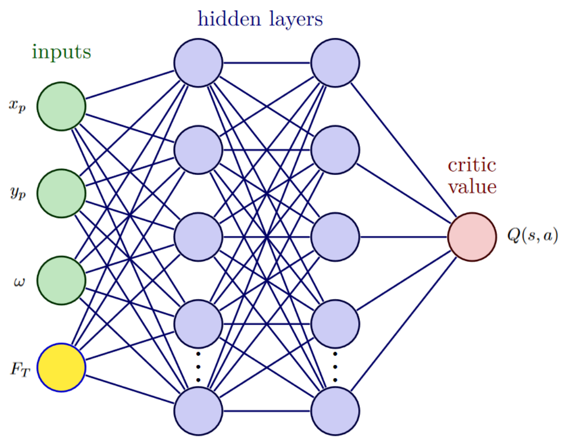


Figure 4.4: Critic network architecture for hinged pendulum environment

In order to verify the code, its performance was measured against a Stable Baselines (SB) implementation of SAC, which is a library that contains implemented reinforcement learning algorithms which are easy to interface with the environments. Using the default hyperparameter values from stable baselines on the author's code yielded similar results with the hinged pendulum being learned in between 20 to 35 episodes as seen in Figure 4.5 and the inverted pendulum in between 200-300 episodes. The differences in the plot can be attributed to a combination of differences in implementation, the stochastic nature of SAC agents as well as the known non-deterministic nature of RL [46].

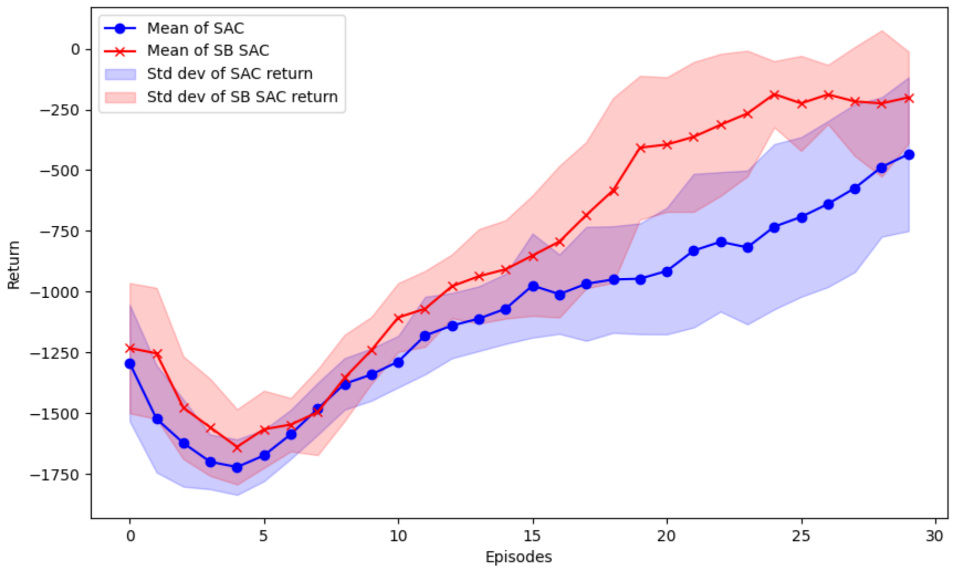


Figure 4.5: SAC comparison with SB SAC on hinged pendulum environment using 30 episode averages over 40 trials

The one hyperparameter that had to be found via trial and error to get the plot above was the entropy coefficient κ as the Stable Baseline implementation uses an automatic entropy tuning procedure as suggested by Haarnoja et al. [47] which in the end was not used in the author's code as it seemed to worsen the performance for the inverted pendulum. So, for verifying the author's SAC implementation worked by comparing its learning performance to SB SAC, $\kappa = 0.2$ was used. Section 4.4 touches on some other methods of verification that were used.

The next step was to perform a hyperparameter optimisation, for which the Optuna python library was used. The hyperparameters and the optimal values found for the hinged pendulum and inverted pendulum agents can be seen in Table 4.1 and Table 4.2 respectively. The tables are separated into three different sections, the first showing the hyperparameters that were fixed and copied from Chen et al. [5] in order to simplify the hyperparameter combinatorial space. The second section depicts the hyperparameters which were optimised using the standard SAC, this was done to ensure the SAC agent was proficient at the task. Note that the buffer size was not included in this section because, due to REDQ algorithm updating several times per step, it would follow that it be more sensitive to the buffer size. Therefore, it was moved to the third section. The third section groups the hyperparameters that were optimised using the REDQ extension of SAC while keeping fixed the values from the first two sections. For a more detailed description of how these values were found see Section 4.3.

Table 4.1: Hyperparameters hinged pendulum for SAC/REDQ agents

Parameter	**Value**
Optimizer[1]	Adam [48]
Nonlinearity	ReLU
# hidden layers	2
# of units per hidden	256
Learning rate[2]	7.5×10^{-4}
Discount factor	0.999
Temperature coeff	0.185
Target smoothing coeff.	0.009
Batch size	256
Buffer size[3]	0.4M
# of critics	10
Subset size	7
UTD ratio	8

[1] From Chen et al. [5]
[2] Group found from SAC hyperparameter study.
[3] Group found from REDQ hyperparameter study.

Table 4.2: Hyperparameters inverted pendulum for SAC/REDQ agents

Parameter	**Value**
Optimizer[1]	Adam [48]
Nonlinearity	ReLU
# hidden layers	2
# of units per hidden	256
Learning rate[2]	6.6×10^{-4}
Discount factor	0.986
Temperature coeff	0.185
Target smoothing coeff.	0.007
Batch size	128
Buffer size [3]	1M
# of critics	5
Subset size	3
UTD ratio	10

[1] From Chen et al. [5]
[2] Group found from SAC hyperparameter study.
[3] Group Found from REDQ hyperparameter study.

With hyperparameter tuning completed, the next step was to actually evaluate and compare the sample efficiency of SAC against the REDQ extended version of it. The metrics for sample efficiency were defined as seen in Equation 4.5, Equation 4.6 for the hinged pendulum and the inverted pendulum respectively based on the return successfully trained agents receive. In these μ_k represents the running mean of the next k episodes. Equation 4.5 is defined in terms of episodes (ep). Each episode is guaranteed 200 interactions with the environment so it holds the same amount of information as number of interactions or steps. Equation 4.6 is also defined using episodes (ep) but note that here the episode lengths now vary depending on how well the pole is balanced (due to the termination condition). This could be considered more aptly as the episodic efficiency, which indirectly points to the sample efficiency as it is ultimately trying to capture the minimum number of episodes the agent experiences before learning the task.

$$J_{\text{hinge}}^{\text{eff.}} = \min |ep_i| \text{ s.t. } R(\mu_5(ep_i)) > -250 \tag{4.5}$$

$$J_{\text{inv}}^{\text{eff.}} = \min |ep_i| \text{ s.t. } R(\mu_5(ep_i)) = 1000 \tag{4.6}$$

To test the efficiency of the sample of a REDQ agent compared to a SAC one, the best performing hyperparameters seen in Table 4.1 and Table 4.2 were used to construct the learning curves. Figure 4.6 shows the comparison of performance between the two algorithms, where the blue line and the contour consist of the average and standard deviation of 40 SAC trials while the red line and the contour represent those of the 24 REDQ trials. The mismatch in the number of trials and the relatively low number of episodes has to do with the computation time difference between the algorithms, with REDQ taking substantially longer due to a higher UTD. This implies more updates to the network per timestep, which is the most computational part of the training. The performance of the REDQ agents are significantly better and with less variance in terms of sample efficiency, narrowing in on the -250 requirement very fast starting at around episode 7 and staying there for the rest of the episodes. The SAC performance is more volatile but on average touches the -250 goal within 17 episodes but with a lot more fluctuation around -250, which continues to taper off as the number of episodes increases.

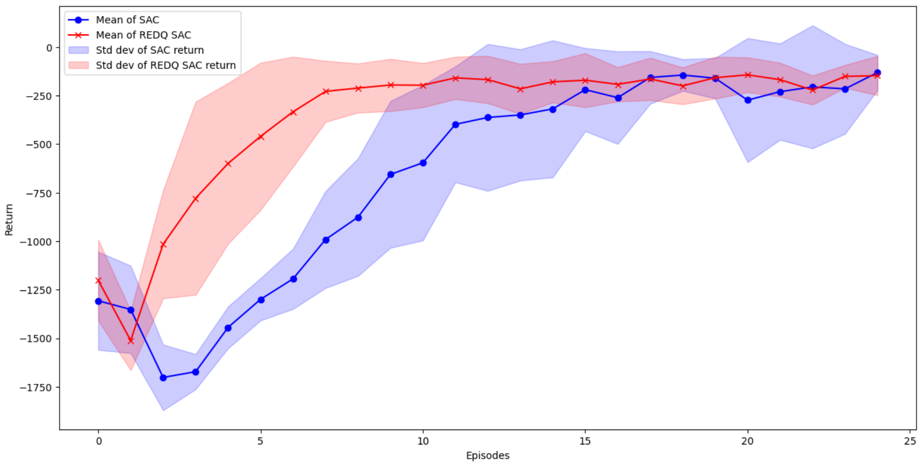


Figure 4.6: Experiment - Hinged pendulum learning curves of SAC and REDQ agents

The training curve comparison between the SAC and the REDQ agent for the inverted pendulum can be seen in Figure 4.7. For the SAC agennts, the blue solid line represents the mean and the shading the standard deviation across 40 trials. The red solid line and shading represent those of the REDQ agents over 20 trials (different due to computation time). It is important to note that since in the inverted pendulum environment the episodes are not of a fixed length, the training curve will aggressively skew towards the better performing agent because it will see more training steps per episode. This means that in the 40 episode performance recorded for each agent, the REDQ agent was able to train substantially more as it learned to avoid termination faster. An estimate of the mean aggregate training steps taken by each agent can be found by summing the mean episodic returns since the sparse reward function rewards a point for every timestep that the episode is not terminated. Over these 60 episodes, due to this correspondence between reward and training steps for the environment SAC trains on average over 562 steps while REDQ does so over 20663 steps. As a result, the training curve overwhelmingly favours REDQ which forks the SAC curve at about episode 22 where it begins to consistently avoid the termination condition.

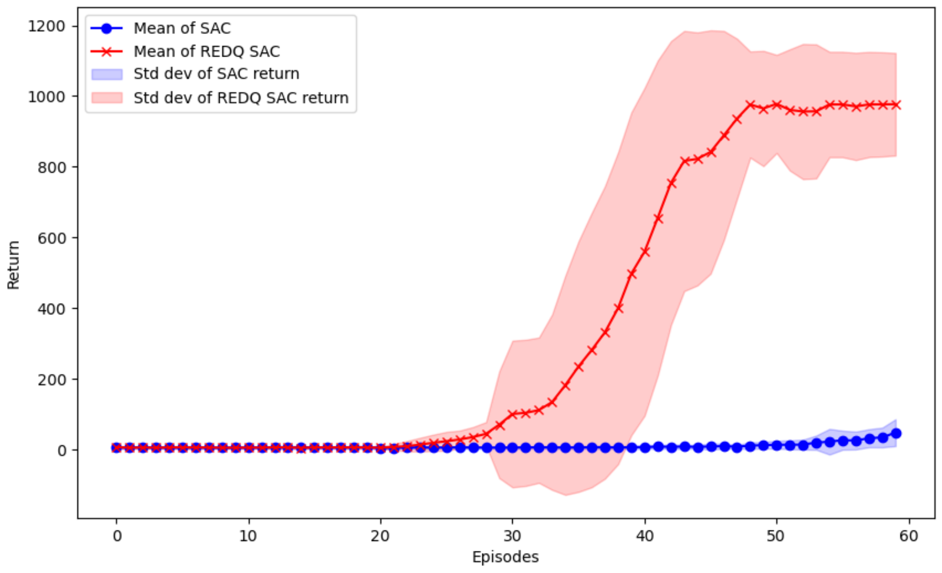


Figure 4.7: Experiment - Inverted pendulum learning curves of SAC and REDQ agents

4.3 Hyperparameter optimisation

The hyperparameter optimisation was performed using the Optuna python library with up to eight trials being carried out in parallel. The search space for the hyperparameters can be seen in Table 4.3. Once again, the first group of hyperparameters were optimized first for SAC and kept fixed when optimising for the second group's REDQ hyperparameters. The first group consisted of mostly type float parameters (these can be freely sampled between the given bounds) with some categorical parameters as well (these have a fixed number of possible choices). The second group consisted fully of parameters which were either integers or categorical. As a result of both being diescrete their parallel coordinate plots (as seen in Figure 4.9 for example) look different relative to the floats.

Table 4.3: Hyperparameter optimisation search space

Parameter	Value Range
Learning rate[1]	[1E-3, 1E-5]
Discount factor	[0.5, 0.999]
Temperature coeff.	[0.1, 0.5]
Target smoothing coeff.	[0.001, 0.01]
Batch size	64, 128, 256
Buffer size	0.2M, 0,4M, 0.6M, 0.8M, 1M
# of critics	[2, 10][2]
Subset size	[2, # of critics]
UTD ratio	[2, 10]

[1]The learning rate for the actor and critic are both the same
[2]The upper bound for number of critics in the inverted pendulum was changed to 6 to reduce computation time.

The sampling from these ranges was done through a process called Tree-structured Parzen Estimator (TPE) sampling [49] which is a Bayesian non-parametric model based sampling technique that continuously uses the observed performance information from trials to refine a probabilistic model between the hyperparameters and the objective to select future hyperparameters in an informed manner (relative to for example random search). For all the trials, no seeds were fixed as hyper-optimising for a seed is not the goal. The objective function was defined as the average reward across the number of episodes per trial and the aim was to maximise it. For the hinged pendulum environment, the first group of parameters yielded the parallel co-ordinate plot seen in Figure 4.8 across 100 trials of 30 episodes (with each episode lasting 200 timesteps) each using SAC.

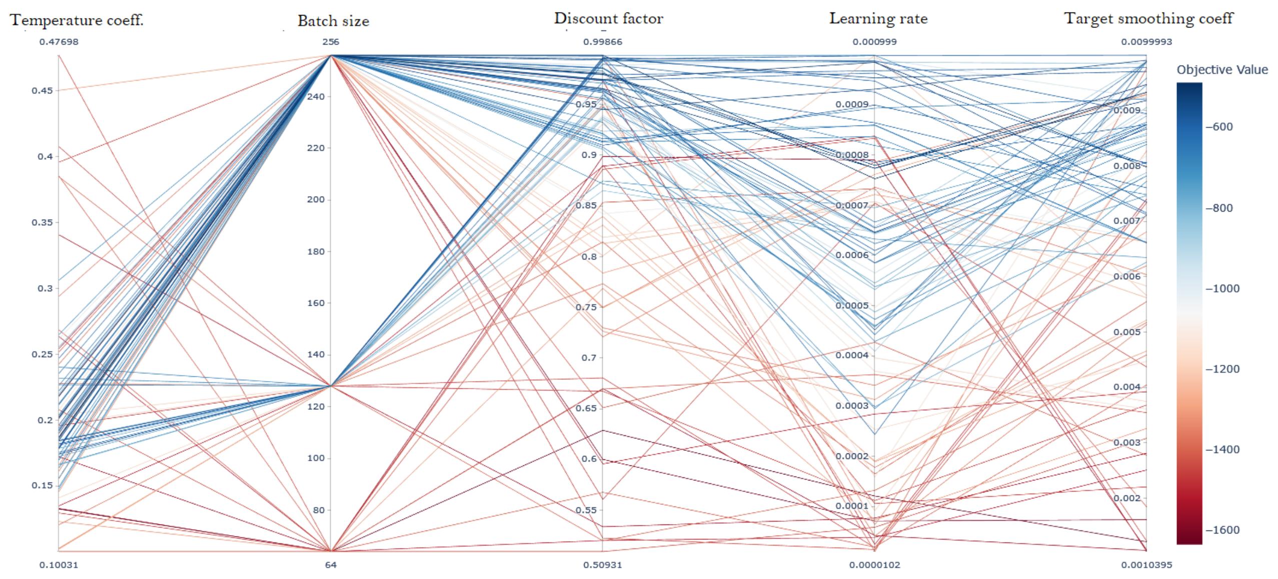


Figure 4.8: Hinged pendulum SAC hyperparameter parallel co-ordinate plot

There is large volatility in the objective value for these trials. This is expected as RL agents are quite sensitive to some hyperparameter choices and this is the initial sifting through the hyperparameter space. Notice how the lines cross freely across the parameter columns for float type parameters whereas they cross them discretely for categorical (batch size). A high batch size, discount factor, learning rate and target smoothing coefficient seem to perform well in the hinged pendulum task. The second group of parameters yielded the parallel co-ordinate plots seen in Figure 4.9 using 70 trials across 20 episodes (200 timesteps per episode) each using the extended REDQ implementation. The motive behind doing less trials and episodes for the REDQ version is because the computation time is much longer than the normal version due to the high number of updates. Notice how all the objective values associated with high UTD ratios are higher.

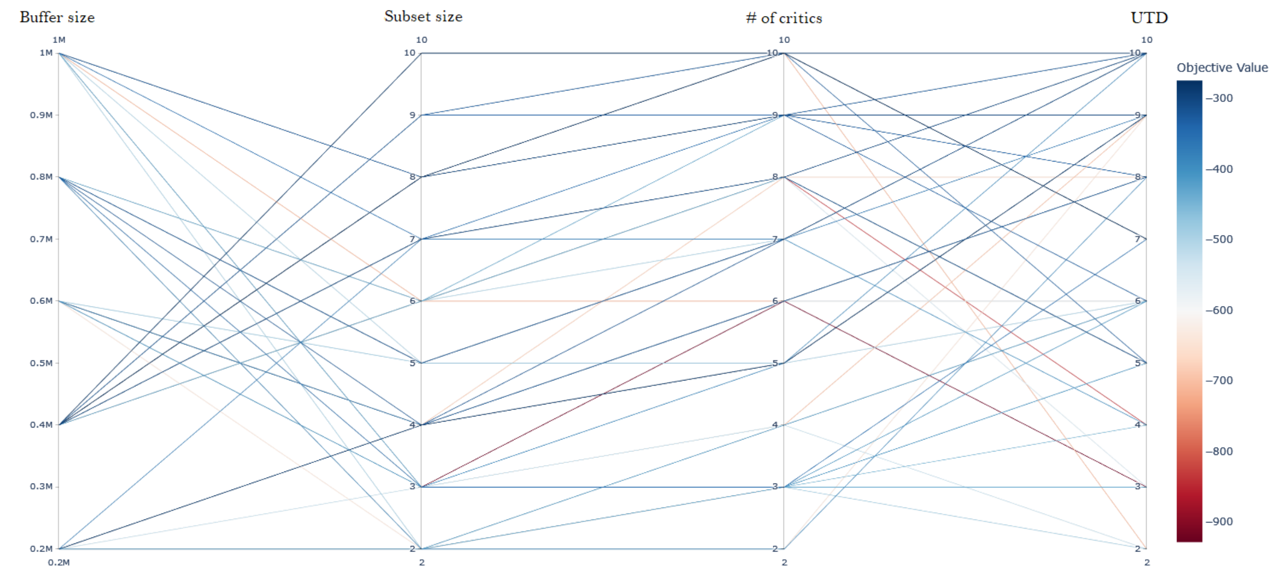


Figure 4.9: Hinged pendulum REDQ hyperparameter parallel co-ordinate plot

The parallel co-ordinate plot for the inverted pendulum using SAC agents can be seen in Figure 4.10, which spanned 200 trials each training for a total of 20000 timesteps. There is a clear trend for good hyperparameters as the best results are clustered with high discount factors, learning rates and target smoothing coefficients. Comparing this plot with that of the SAC hinged pendulum confirms that the best performing hyperparameters are dependent on the environment of the task.

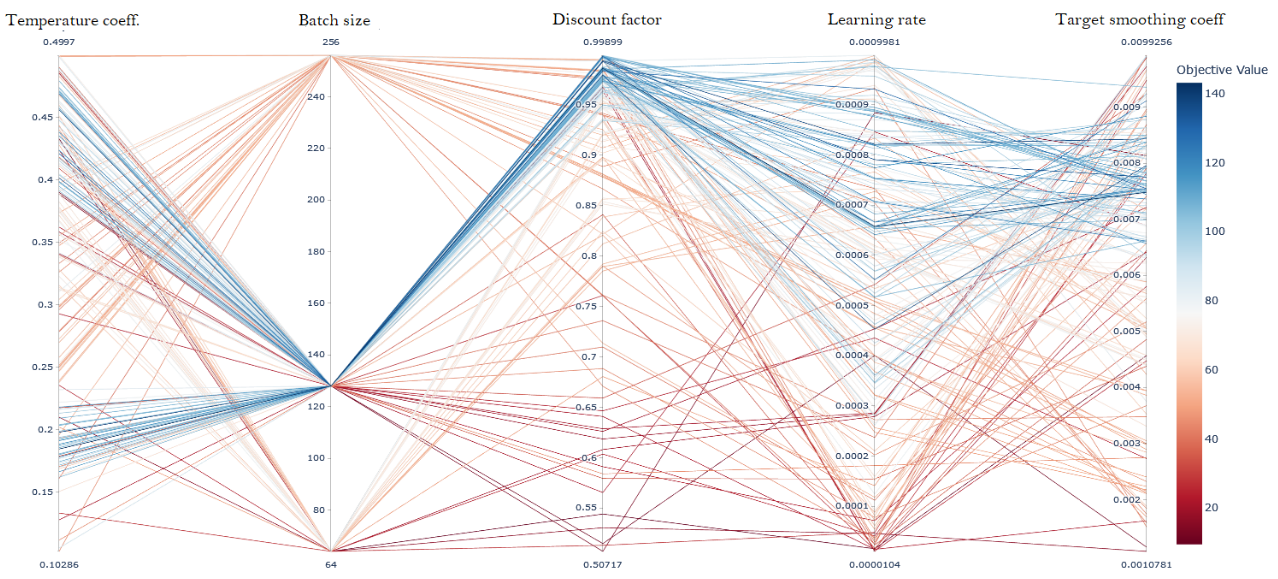


Figure 4.10: Inverted pendulum SAC hyperparameter parallel co-ordinate plot

Figure 4.11 shows the inverted pendulum performance of REDQ agents across 70 trials that trained for a total of 20000 timesteps each. From here, one can again see that higher values of UTD seem to be performing very well relative to those with lower UTD.

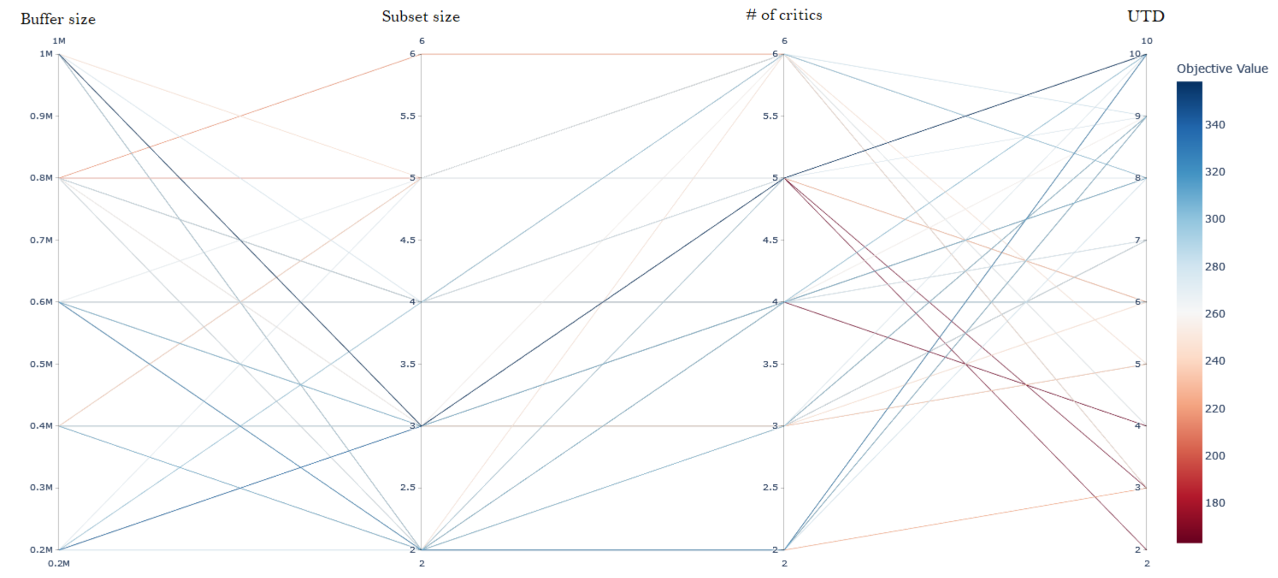


Figure 4.11: Inverted pendulum REDQ hyperparameter parallel co-ordinate plot

Hyperparameter importance

Now that visualisations have been given for the combinatorial space within the bounds in Table 4.3, a natural question to ask is "how important is each of the hyperparameters to the overall objective value?". This is captured in the hyperparameter importance metric suggested by Hutter et al. [50]. It's a metric that seeks to quantify the importance of each hyperparameter of a black-box model (the trial observations) by approximating it with a learned random forest model and then marginalising the variance observed with respect to each hyperparameter. The hyperparameter importance is defined between $[0, 1]$ with higher values corresponding to hyperparameters whose change highly affects the objective performance. A note should be made that the hyperparameter importance is of course dependent on the search space defined as if one picks to search within a narrow "good enough" region in one parameter and a wider region in another with variable performance then the latter will be considered more important.

The hyperparameter importances for the SAC and REDQ in the hinged pendulum environment can be seen in Figure 4.12 and Figure 4.13 respectively. For the SAC optimisation study the target smoothing coefficient (ρ), learning rate and discount factor (γ) were found to be the most important. In the case of the REDQ study it was found that the UTD was the most important followed by the buffer size. This is in agreement with the results in Chen et al. [5], where a UTD of 20 in order to achieve their best sample efficient performance.

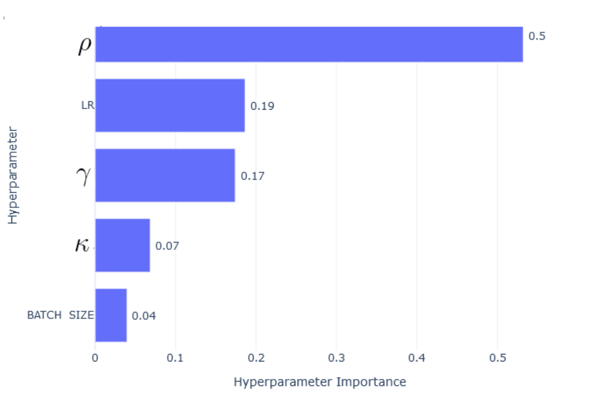


Figure 4.12: SAC hinged pendulum parameter importance

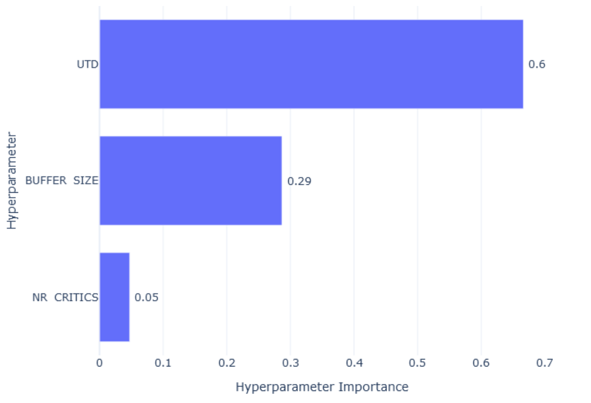


Figure 4.13: REDQ SAC hinged pendulum parameter importance

Figure 4.14 and Figure 4.15 show the hyperparameter importance for the SAC and REDQ studies respectively. There is an immediate difference in which hyperparameters are most important compared to the hinged pendulum once again, confirming that the importance of hyperparameters varies depending on the task. In the case of the SAC agent the discount factor (γ) and learning rate were found to be the most important while UTD was the only real important hyperparameter in the REDQ study. The general trend across the hyperparameter optimisation is that REDQ is very sensitive to the UTD and that higher UTD seems to perform better (at the expense of more computation time).

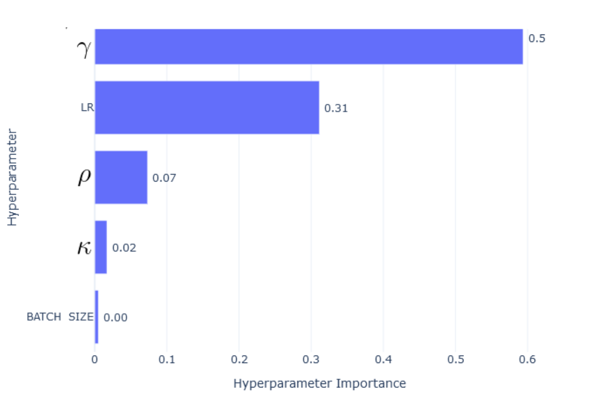


Figure 4.14: SAC inverted pendulum parameter importance

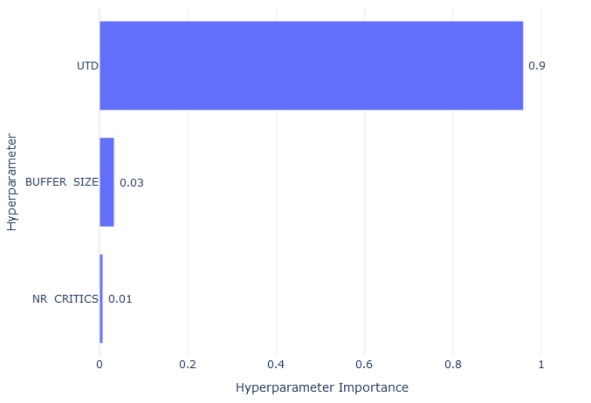


Figure 4.15: REDQ SAC inverted pendulum parameter importance

Final choice of hyperparameters

The chosen hyperparameters in Table 4.1 and Table 4.2 were picked with consideration to the trial hyperparameter performance, the parallel co-ordinate plots and the hyperparameter importance. Meaning that if the best best trial was in line with the story from the parallel co-ordinate plots, then they were chosen. If other trials performed similarly but with simpler hyperparameters (computation wise such as less number of critics) these were chosen as long as their hyperparameter importance was low (less than 0.1). It is important to mention this because in isolation the trial performance is not a sufficient criterion for choosing hyperparameters. This is because it could be the result of favourable environment initialisations and choices of hyperparameters were not repeated in this hyperparameter study for several different environment seeds.

4.4 Verification

As previously discussed, one way in which the code was verified was using the SB library to compare performance. However, other methods of verification were also used ranging from print statements at a low level to check everything was working under the hood, up to more system scale tests. This section will focus mostly on the latter and the policies of the actor will be used as evidence that the agent is learning good policies. These policies used the best found hyperparameters from the hyperparameter optimisation (except on the policy for the REDQ inverted pendulum where a lower UTD was used to reduce computation time). The verification using the hinged pendulum will be given in Subsection 4.4.1 and the inverted pendulum in Subsection 4.4.2.

4.4.1 Verification using hinged pendulum

The best way to interpret the following policies is to keep an eye on Figure 4.1 for the sign convention and keep in mind that for the following policies the x_p and y_p have been replaced by $-\pi \leq \beta_p \leq \pi$ using $x = 1 \times cos(\beta)$ (the pendulum has a length of one) to reduce the state dimensions to two. In order to construct these following policy diagrams, the agents were trained until the task had been learned and then the actor network was used to generate the action values per given states.

Looking at Figure 4.16 there are a few things that can be seen to that indicates the agent learned a good policy. Firstly, looking at the corners of the state grid the action follows the direction of the angular velocity ω of the pendulum. This makes sense because both the horizontal extremes represent the pendulum pointing down and so the easiest way to return to the upright position would be to ride in the direction of ω and can be seen by both top corners being red (where ω is positive) and the bottom corners being blue (where ω is negative). Secondly, starting in the centre of the picture and solely moving up vertically would indicate a scenario where the pendulum is at the desired position but with an increasing positive ω, the resultant torque action is to counteract the positive angular velocity which is what's necessary to keep upright equilibrium. The opposite also holds moving down vertically so it is consistent. Lastly, if starting at the centre and moving only horizontally then there is a small margin in which the agent tries to fight this angular displacement by applying an opposite torque. After this margin ends the agent seems to apply full torque in the direction of the displacement meaning it has figured out a threshold at which it's actually better to give the pendulum a push in order to catch the equilibrium on the other side than try to fight the angular displacement.

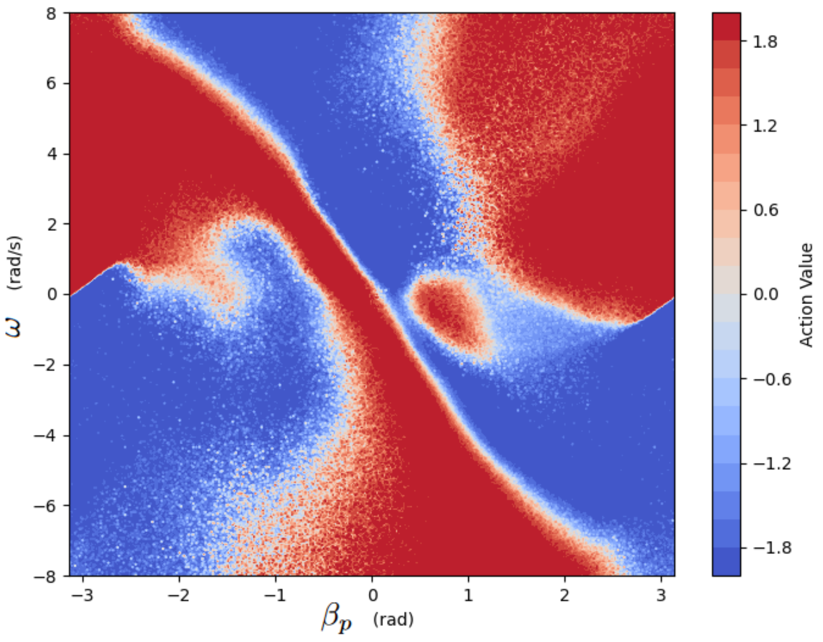


Figure 4.16: SAC hinged pendulum policy after 60 episodes

The policy for REDQ SAC is plotted in Figure 4.17. The same arguments to its validity hold as discussed above, however notice there are more blemishes and irregularities in this policy. These are attributed to the fact that it was mapped after only 20 episodes and also consider that these problems do not have a unique policy solution.

Instead, there is a whole family of non-identical policies which are good enough to solve the hinged pendulum environment.

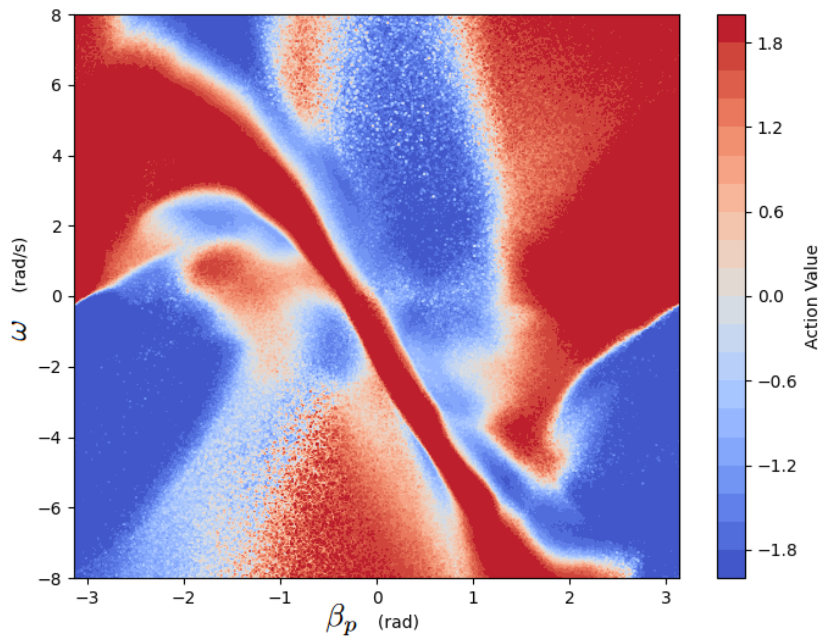


Figure 4.17: REDQ SAC hinged pendulum policy after 20 episodes

4.4.2 Verification using inverted pendulum

The policies for the inverted pendulum were also visualised. In order to be more easily understood it's best to look at the diagram and understand the sign convention discussed in Subsection 4.1.2. For these following policies, the agent was trained until the task was learned. The policies then were plotted using the actor network.

Starting with the SAC policy seen in Figure 4.18, the easiest ones to interpret are those plotting x_{cart} (the bottom three sub-figures) with any of the other variables as x_{cart} is effectively irrelevant to the goal of keeping the pendulum upright. From this, it can be seen that when the angular velocity and pole angle are varied on the bottom left and right plot the action follows the input that would be needed to counteract the rotation. Note that the bottom middle figure fixes both the pole angle and angular velocity meaning that it is essentially at the goal state (perfectly upright with no angular velocity) so in this condition the agent doesn't care too much what the action is as it has learned to address when it isn't stable. For the top sub-figures, the top middle figure is the easiest to interpret as the agent has learned to prioritise counteracting the angular velocity. This is what determines where the pole will be in the future, so it only inputs actions in a way to oppose the angular velocity of the pole regardless of the angle of the pole. In the top right sub-figure it shows that when there is only some angular deflection (without angular velocity) the agent tries to fight it. However, it recognises that for certain deflection and velocity combinations, it doesn't need to input anything as the cart is already moving in such a way that will balance the pendulum and sometimes even pushes against this vertical restoring velocity of the cart (seen in the wavy patterns) in order to arrive at the vertical at a standstill. The same idea holds true for the top left figure.

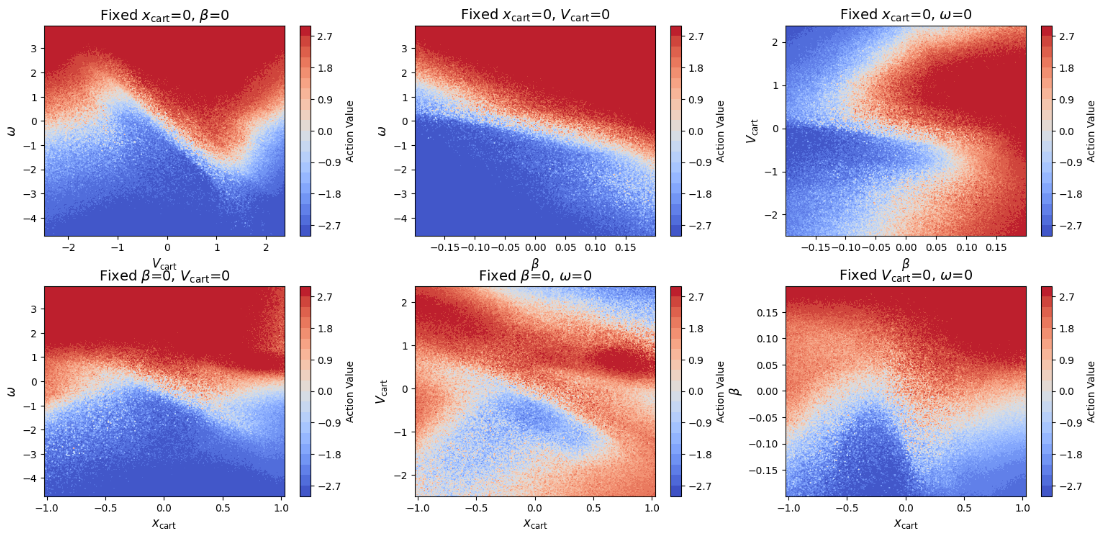


Figure 4.18: SAC inverted pendulum policy study after 200 episodes

The policy for the REDQ variant is plotted in Figure 4.19. As can be seen, the majority of the plots have the same look with some small differences to the decision boundaries. The two plots that are the most different are the top right and bottom middle sub-figures. The differences in the top right plot can be attributed to simply lack of experience for the policy as it took a smaller number of episodes for the REDQ agent to learn the task it wasn't exposed to as many starting conditions as the SAC one. The bottom middle plot is mostly uninteresting as this one fixes the pendulum at the goal state so policy differences in this setting are not of much concern.

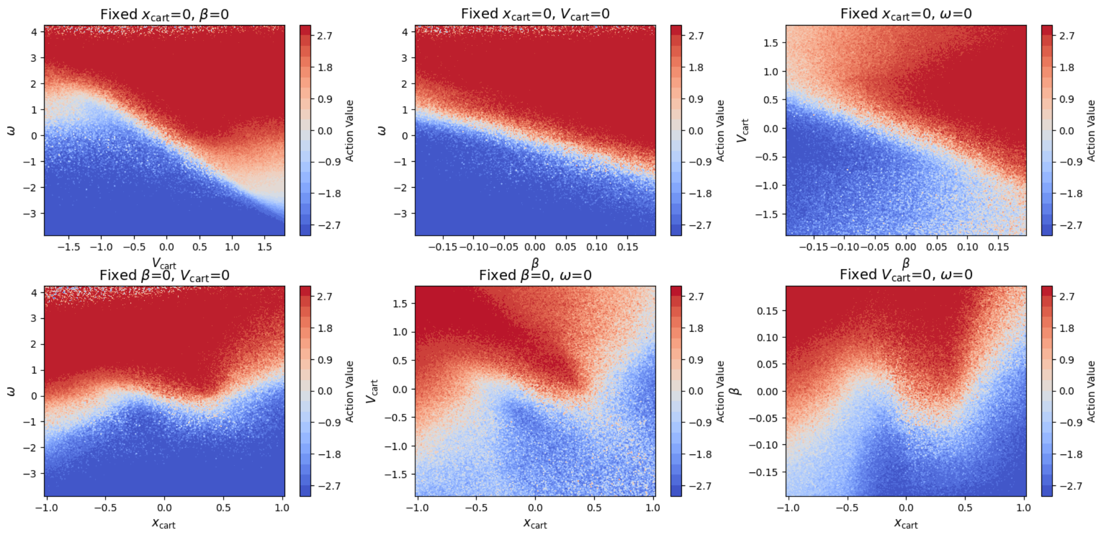


Figure 4.19: REDQ SAC inverted pendulum policy after 50 episodes

Chapter 5

Conclusion

In this literature study, the idea of improving the sample efficiency of current reinforcement learning techniques in flight control applications was discussed and motivated, relevant literature was recapped and a preliminary analysis was conducted. The relevant literature aimed to give an overview of the concepts in RL and address RQ1.

Q1 How can reinforcement learning with a focus on sample efficiency be used for flight control applications?

1. How can RL be used to solve control problems?
2. What are some relevant performance metrics to keep in mind when developing RL for flight control purposes?
3. What is a suitable methodology to tackle sample efficiency improvement?

The first two questions have been addressed. Firstly, reinforcement learning can be used to solve control problems formulated as MDP. In the field of flight control, several applications were mentioned such as the flying of a quadrotors drone with a PPO algorithm and using an IDHP agent to learn a fault tolerant pitch control task online. Secondly, there are several relevant metrics to keep in mind when developing a RL solution in flight control among which are safety, sample efficiency and the explainability of the solution. Abstract mathematical definitions are presented as the precise definitions are problem dependent. The last question remains partially unanswered as though a lot of the relevant literature points to directions in which tackling sample efficiency can be improved, since the thesis exercise has not yet been performed the full details are still missing.

The preliminary analysis looked to address RQ2.

Q2 How can a preliminary analysis investigating sample efficiency be implemented on benchmark environments?

1. What learning environments will be used to test the methodology?
2. What baseline algorithms will be used in the preliminary analysis for comparison?
3. How will the sample efficiency of the preliminary analysis be computed?

The questions will now be addressed in order of appearance. Firstly, the preliminary analysis was implemented in two environments the “Pendulum-v1” and “InvertedPendulum-v4” from the Gymnasium and Mujoco libraries by interfacing with their environments. These two were chosen because one of them was simple to learn for the agents which was good for testing and verification purposes while the other posed a more complex task. Secondly, the baseline algorithm that was used during the preliminary for comparison was SAC and it was chosen because it already had good base performance and REDQ [5] had hyperparameters specific to the SAC implementation for reference. Lastly, environment specific sample efficiency metrics were defined in order to be able to compare the REDQ algorithm’s performance relative to the baseline SAC.

The literature recap also covered some of RQ3.

Q3 How does the proposed methodology perform on the Cessna Citation PH-LAB Model?

1. How can the RL algorithm be interfaced with the PH-Lab model?
2. What algorithms will be used as comparisons to the proposed methodology?
3. How will the proposed methodology improve sample efficiency?
4. What are the safety implications for the proposed methodology?

Addressing them by order of appearance. Firstly, RL algorithms can be interfaced with the PH-LAB via the state space description of the model which can be used to define the agent, its actions and the environment. The full details of how this would be done is not described in detail as it's yet to be implemented but the general description of how it would work is discussed. Secondly, the comparisons to the proposed methodology have yet to be decided (also discussed at the end of this chapter as possible directions) but two examples are given one for an online RL IDHP agent for fault tolerant control and other being an offline attitude and altitude controller using SAC agents. There's also the possibility of just comparing its performance with a control SAC agent. Thirdly, the gains in sample efficiency are likely to come from the power of the REDQ extended algorithms which as seen in the preliminary analysis are more sample efficient than standard implementations with UTD ratios of one but as the experiment has yet to be performed this question has not been fully answered. Lastly, the safety implications of the proposed methodology is not addressed as this is something that can only be discussed in the context of the experiment environment under safety criteria which have not yet been set.
For continuing into the thesis exercise, the following directions are given as ideas to what the author currently has in mind:

(A) The offline sample efficiency of standard SAC and REDQ will be compared for a pitch control tracking task like Figure 3.2

(B) Compare offline sample efficiency in Dally [37] experiment by extending their SAC implementation with REDQ

(C) The online sample efficiency will be tested using the online learning experiment setup proposed by Chan [36]:

- is REDQ sample efficient enough to learn a task in an online setting?
- if not, what can be changed to the experiment such that it's sample efficient in an online task? (Increase warmup phase or change the amount of information in warmup phase)

Part III

Additional Results

Chapter 6

Hyperparameter Tuning

Hyperparameter choice plays an important role on the agent's learning, to ensure the comparison between agents was fair a hyperparameter search was conducted to find the best combination for SAC and REDQ. This chapter will discuss the search procedure for the policy smoothness hyperparameters in Section 6.1, how the hyperparameter tuning was performed in Section 6.2 and the final choice of hyperparameters in Section 6.3. The choice of keeping the smoothness and other hyperparameter tuning separate was made to first establish a smooth starting point around which the agents could then be tuned.

6.1 Policy smoothing - CAPS hyperparameters

To ensure that the policy of the agents was smooth for use on real-life applications (that is, to prevent bang-bang control), Conditioning for Action Policy Smoothness (CAPS) [51] was used to smoothen the policy during training. In essence, CAPS introduces two regularisation terms to the actor loss objective function: one for the spatial smoothness (to ensure that actions near similar states are not too different) and one for the temporal smoothness (to ensure two sequential actions are not too different from one another). Each of these regularisation terms has their own coefficient in the objective function which are hyperparameters, denoted η_s and η_t respectively.

Two figures are provided to illustrate the difference between agents that learn with CAPS and those that do not. Figure 6.1 shows an agent that's learned a pitch tracking task without policy smoothness. While the tracking performance is accurate, the elevator control exhibits highly volatile bang-bang behavior. Such abrupt control actions are undesirable for policies intended for physical systems due to concerns over safety, energy efficiency and wear caused by friction.

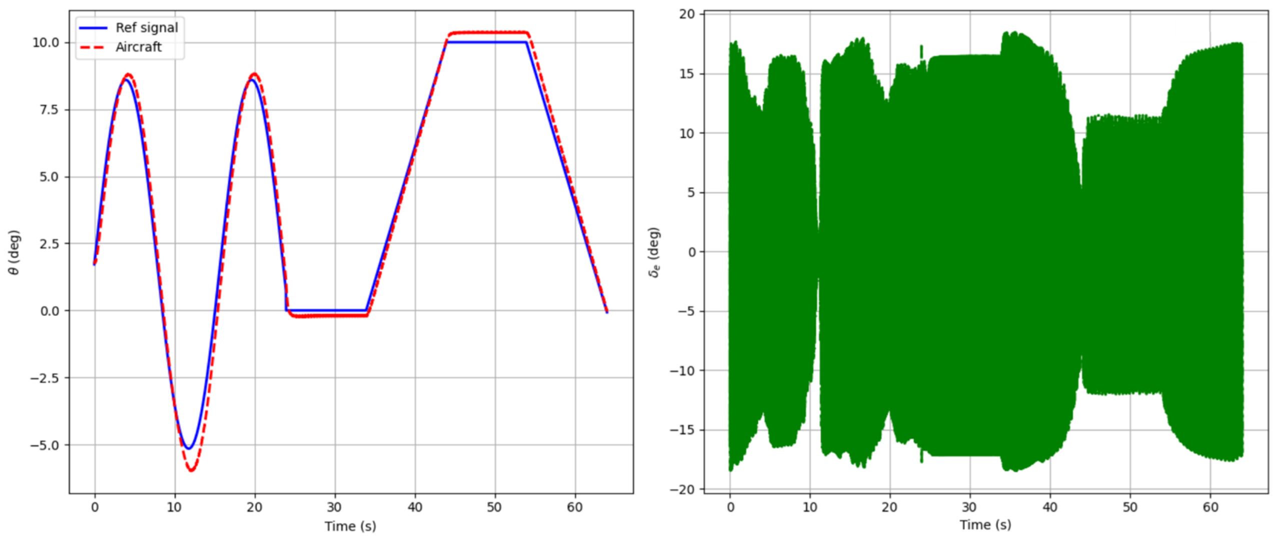


Figure 6.1: Pitch tracking by SAC agent without CAPS: pitch (left) and elevator action (right)

Figure 6.2 shows the result of training an agent with CAPS. Here, the tracking performance is also good and the actuator exhibits smooth behaviour.

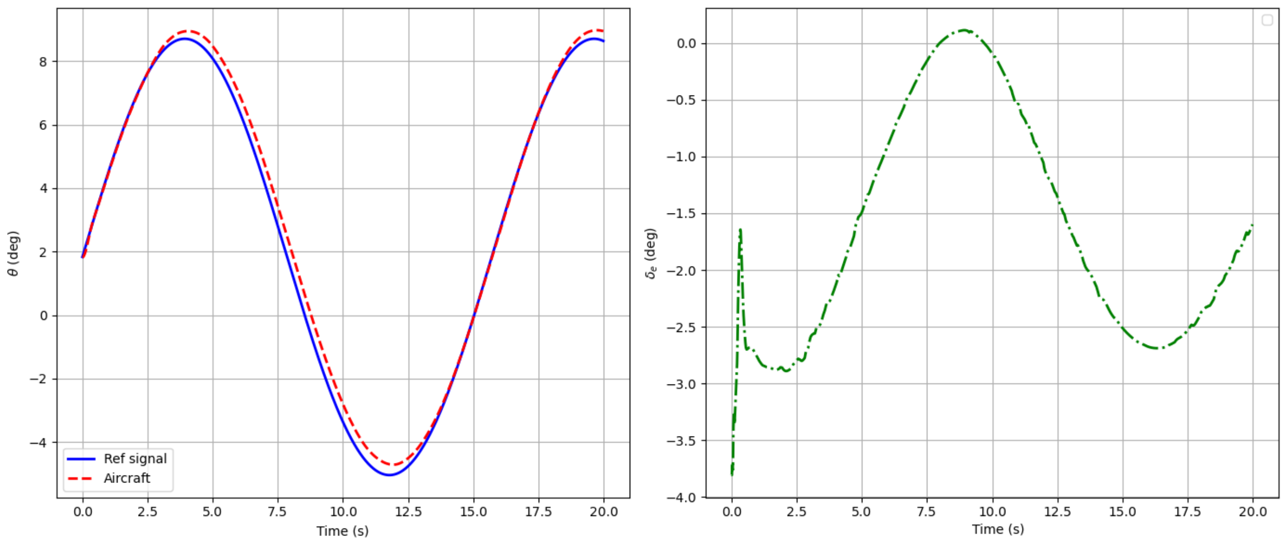


Figure 6.2: Pitch tracking by SAC agent CAPS $\eta_t = 15, \eta_s = 5$: pitch (left) and elevator action (right)

The hyperparameter values for η_s and η_t were found experimentally by changing their values then looking at the actuator output once the learning was finished. This was repeated until a suitable combination of η_s and η_t were obtained based on how smooth the time series visualisations were. For the pitch and roll tracking tasks, it was found that the agent's policy was most sensitive to the temporal smoothness coefficient. For these, a η_t value of 15 yielded great results while the spacial smoothness coefficient seemed to have a smaller impact on the smoothness of the agent's actions with a η_s of 5 yielding good results. For the biaxial control task, $\eta_s = 30, \eta_t = 60$ yielded mostly smooth results.

Although these hyperparameters often lead to smooth policies for agents, the number of training steps needed to become smooth is not always the same. Agents will often appear to have learned the task and show good results in terms of return, but the policy will still be volatile. This is because agents converge to smooth policies at different rates. Creating a metric to track smoothness is non-trivial as a threshold needs to be set to classify what is considered smooth and what is not during any evaluation. Therefore, it is recommended to always visually check the smoothness of the policies to verify if the agent has converged to an acceptable policy in both return performance and smoothness.

6.2 Learning-related hyperparameter optimisation

For the learning-related hyperparameters, a hyperparameter study was conducted in Optuna using TPE sampling for parameter search in the maximise direction. Each trial consisted of training the agent on a pitch tracking task for 20 episodes each lasting 50 seconds with the trial objective being the mean reward across all 20 episodes. The search space for the SAC hyperparameters can be seen in Table 6.1. Note that some fixed hyperparameters were chosen based on those found by Dally [37] (like the number of hidden layers and neurons per layer). The ranges of the search space were determined by what the author believed to be sensible based on initial manual trial and error attempts.

Table 6.1: SAC Hyperparameter optimisation search space

Parameter	**Value Range**
Learning rate	[1E-3, 3E-5]
Discount factor	[0.9, 0.99]
Target smoothing coeff.	[0.01, 0.3]
Batch size	64, 128, 256

[1]The learning rate for the actor and critic are both the same

The parallel co-ordinate plot for this search can be seen in Figure 6.3. From the plot, it is clear that a combination of a low batch size, high discount factor, medium learning rate and low target smoothing coefficient performs best. Note the hyperparameter search space type affects how the parallel co-ordinate plot looks, with discrete search spaces like batch size showing less variance compared to float search spaces like learning rate.

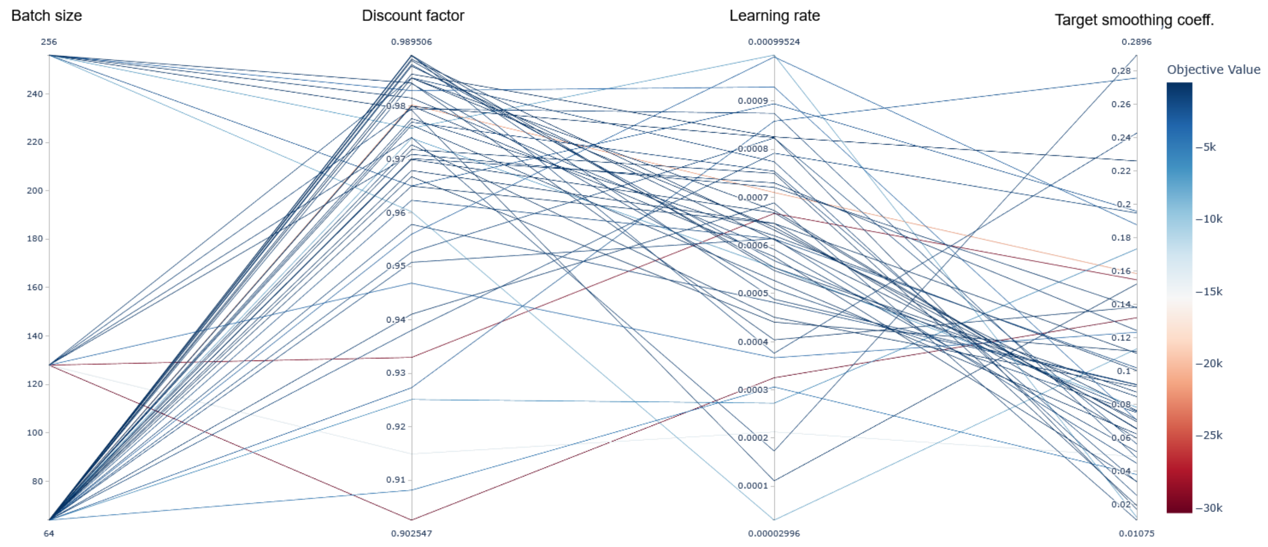


Figure 6.3: Parallel coordinate plot of SAC hyperparameters for the pitch tracking task

The parameter search space for the REDQ agents can be seen in Table 6.2. Here, the upper bounds for the number of critics and UTD were kept low due to computation time considerations. The subset size was omitted and dropout was explored following developments from Hiraoka et al. [52] that showed great results in sample efficiency on benchmarks when these two choices were made.

Table 6.2: REDQ Hyperparameter optimisation search space

Parameter	**Value Range**
Learning rate[1]	[1E-3, 3E-5]
Discount factor	[0.9, 0.99]
Target smoothing coeff.	[0.01, 0.3]
Batch size	64, 128, 256
Buffer size	50k, 100k, 150k, 200k, 250k
# of critics	[2, 5]
Subset size	-
UTD ratio	[2, 5]
Dropout chance	[0, 0.3]

[1]The learning rate for the actor and critic are both the same

The accompanying parallel co-ordinate plot can be seen in Figure 6.4. Here, a high discount factor, high UTD, low learning rate, and low number of critics seem to perform best. The dropout was set to zero after a few trials since it caused the agent to consistently underperform whenever it had a non-zero probability.

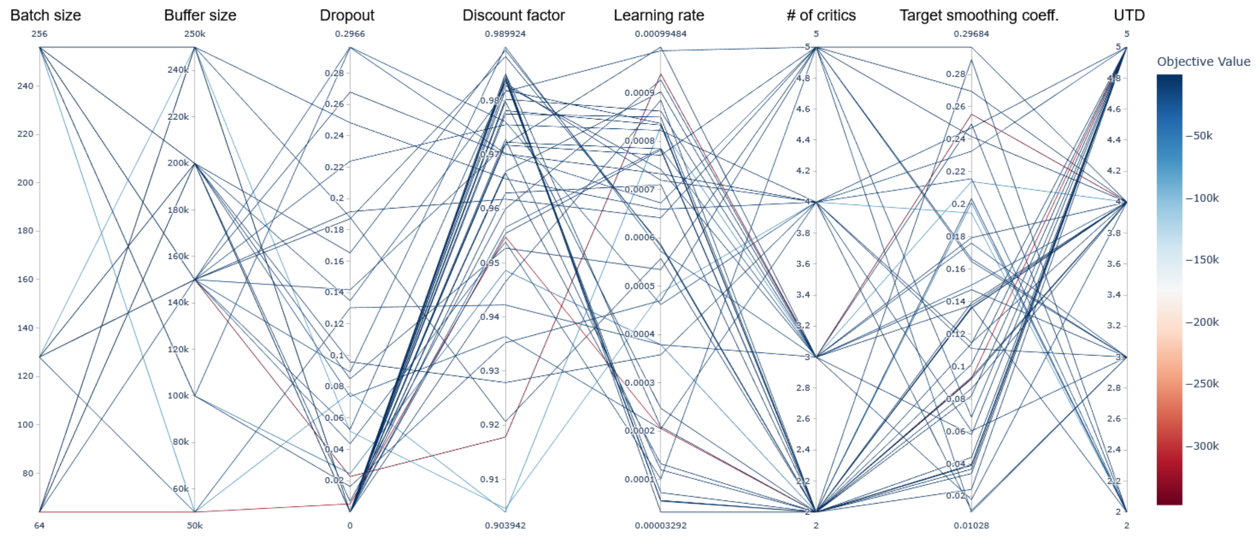


Figure 6.4: Parallel coordinate plot of REDQ hyperparameters for the pitch tracking task

The hyperparameter importance for SAC and REDQ SAC on the pitch task are given in Figure 6.5 and Figure 6.6. These are found automatically through the Optuna hyperparameter study using the analysis of variance procedure outlined by Hutter et al. [50]. In essence, it works by applying non-parametric regression across the observed trials and ranks them based on the effect each hyperparameter has on the variance when marginalised with respect to the objective performance (which was the mean reward across 20 episodes for this study). By definition, the hyperparameter importances sum to one with higher values indicating more important hyperparameters. Note that the importance is based on how the hyperparameter search space is defined, as hyperparameters that explore broader search spaces tend to induce higher performance variance.

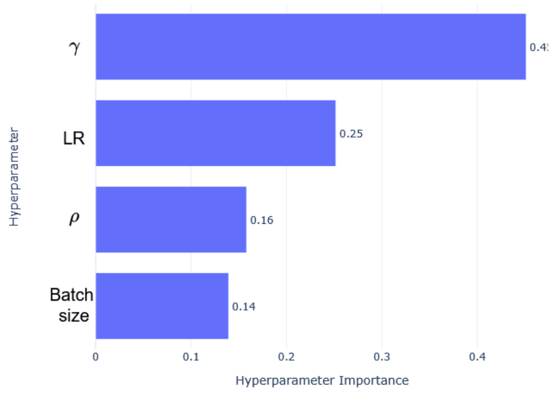


Figure 6.5: SAC pitch tracking task parameter importance

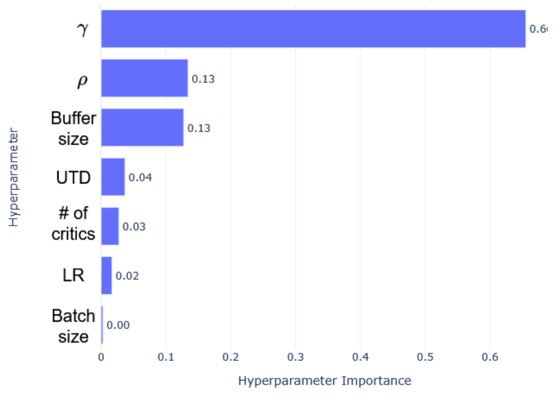


Figure 6.6: REDQ SAC pitch tracking task parameter importance

Each plot tells a different story about the importance of the hyperparamters. However, both agree that the discount factor γ is the most important hyperparameter. One-to-one comparisons of hyperparameter rankings are challenging because one algorithm uses more hyperparameters, which dilutes the importance percentage. The biggest discrepancy between importance is in the learning rate, which the plots show SAC is more sensitive to than REDQ likely due to the higher UTD ratios compensating for slower learning rates. For the REDQ hyperparameters, their importance was generally secondary to that of the base SAC hyperparameters, with the UTD ratio and the number of critics showing roughly equal importance. Dropout was omitted since it was set to zero after initial exploration. However, since non-zero dropout probabilities systematically caused the agents to have deteriorated learning performance when it was non-zero, it likely has a high hyperparameter importance.

6.3 Final hyperparameters

The hyperparameters obtained from the hyperparameter study serve as a good base from which to compare the sample efficiency between agents. The final choice of hyperparameters were found by looking at the SAC parallel co-ordinate plot and the best trials within that study. If the parameters of the best trials matched the trends of the plot, they were chosen. Else, they were picked based on which parameter regions had the most successful trials from the plots with consideration on computing time (such as choosing fewer critics if possible).

In the end, the optimal REDQ agent hyperparameters from the Optuna study were not used for the experiment as they would be an uncontrolled addition to the variance in the sample efficiency performance of the agents. This is because, for finding hyperparamters, the objective of the study was to maximise the mean reward across 20 episodes. This objective does not necessarily translate to better sample efficiency. Instead, to isolate the impact the REDQ hyperparameters have on the sample efficiency, two agents were proposed RED3Q and RED5Q. These agents had a matching number of UTD and number of critics set by the number in the name. RED3Q was proposed as a minimal incremental step to SAC, only increasing the number of critics by one and tripling the UTD ratio. RED5Q constituted a larger step with respect to SAC, adding three more critics and quintupling the UTD ratio. The motive behind keeping the number of critics and UTD ratio the same was mostly due to combinatorial simplicity, as investigating the sensitivity to each of these hyperparameters individually for every task would require even more agents to be trained, i.e. an agent with a UTD ratio of two with three critics and another agent with a UTD ratio of two but with five critics instead to isolate and investigate the effect of number of critics on performance. Table 6.3 and Table 6.4 give the chosen SAC and REDQ hyperparameters respectively.

Table 6.3: Hyperparameters SAC agents

Parameter	**Value**
Optimizer	Adam [48]
Nonlinearity	ReLU
# hidden layers	2
# of units per hidden	64
Target entropy	$-\text{dim}(A)$
Spacial smoothness coeff.[1]	5 | 30
Temporal smoothness coeff.[1]	15 | 60
Learning rate	6.3×10^{-4}
Discount factor	0.99
Target smoothing coeff.	0.076
Batch size	64
Buffer size	50k

[1]Left: pitch or roll. Right: biaxial control.

Table 6.4: Hyperparameters REDQ agents

Parameter	**Value**
Optimizer	Adam [48]
Nonlinearity	ReLU
# hidden layers	2
# of units per hidden	64
Target entropy	$-\text{dim}(A)$
Spacial smoothness coeff.[1]	5 | 30
Temporal smoothness coeff.[1]	15 | 60
Learning rate	6.3×10^{-4}
Discount factor	0.99
Target smoothing coeff.	0.076
Batch size	64
Buffer size	50k
# of critics[2]	3 | 5
UTD ratio[2]	3 | 5

[1]Left: pitch or roll. Right: biaxial control.
[2]For RED3Q and RED5Q respectively.

Note that in the end, a buffer size of 50 thousand was used because all tasks were learned in less than 50 thousand training steps during the actual experiments. The hyperparameters used for the pitch, roll and biaxial tracking agents were the same.

Chapter 7

Time Series Fault Analysis

This chapter will display a selection of time series plots and accompanying discussion for a deeper dive at how agents performed in the different fault scenarios mentioned in the scientific article. Due to the large number of combinations of agents and tasks, only a subset of the plots are presented. The nominal condition behaviour will be discussed in Section 7.1, the quarter actuator efficiency condition in Section 7.2 and the actuator jolt condition in Section 7.3.

7.1 Nominal condition agent behaviour

The biaxial control behaviour for the nominal condition is shown in Figure 7.1. From the figure, it is evident that the agent is able to accurately track the reference signals in pitch and roll simultaneously. Furthermore, at the very beginning of the evaluation the agent initially shows a quick oscillation on both actuators of differing magnitudes that quickly dies down. The motive behind this is not fully understood, however, since the agent starts at the trim condition with zero pitch/roll rate and zero error it is possible that this is simply a scenario that very seldom occurs during training so it's actions are erratic. Looking at the yaw angle, a drift towards a steadily increasing yaw angle is visible. This is expected as the agent offers no yaw control and the model has a pitch damper present, which only mitigates quick oscillations and not a slow drift. The altitude slowly decreases over time since there is no altitude control. Finally, the velocity oscillates around 90m/s due to the auto-throttle present in the model.

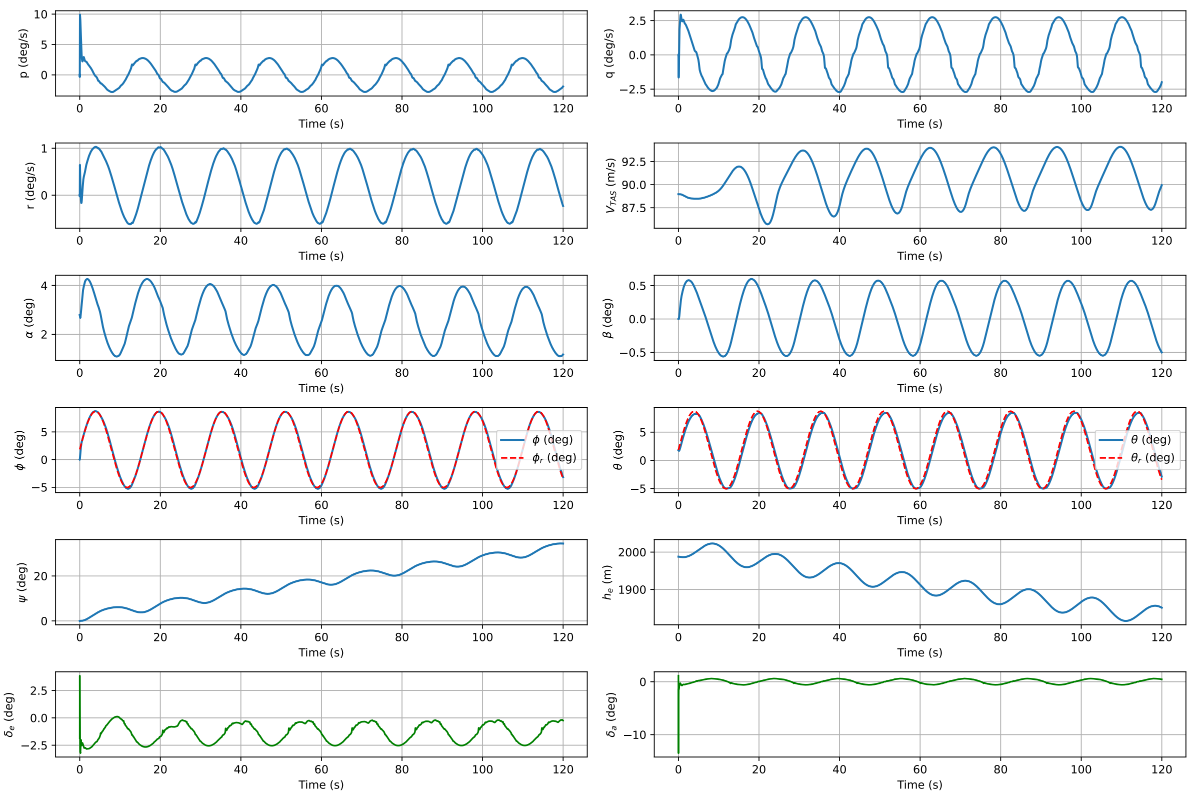


Figure 7.1: Smooth RED3Q nominal condition biaxial control behaviour

The behaviour of the agents is not always smooth as shown in Figure 7.2. Here, the agent seems to have difficulties in both pitch and roll at the bottom of the pitch/roll reference as this is where the volatile behaviour consistently occurs. However, the performance does not change in these volatile regions, further illustrating the problem of ensuring smoothness discussed in Section 6.1 and demonstrating the importance of visualisation accompanied by performance metrics.

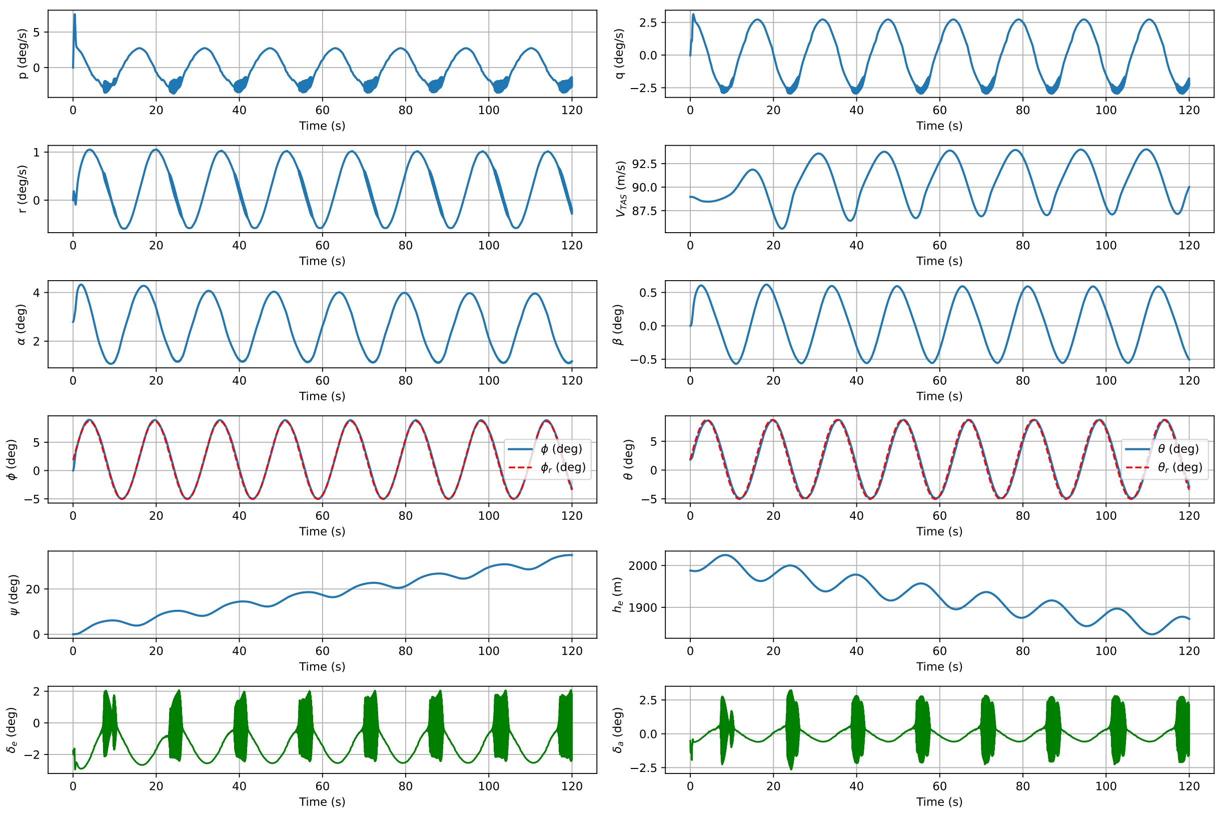


Figure 7.2: Non-smooth RED3Q nominal condition biaxial control behaviour

7.2 Quarter actuator efficiency condition agent behaviour

The roll control behaviour for the quarter actuator efficiency condition is given in Figure 7.3. Here, it can be seen that the agent is able to accurately track the roll reference even with diminished control over the aileron. This is likely tied with the fact that the policy being learned, while effective, is not minimal. This means that the magnitude of the agent actions could still be optimised to be even smaller and more energy efficient. Coincidentally, the diminished actuator actions also alleviates some of the initial erratic behaviour at the beginning of evaluations observed under the nominal conditions. Furthermore, looking at the longitudinal states, there are still oscillations in them due to weakly coupled dynamics.

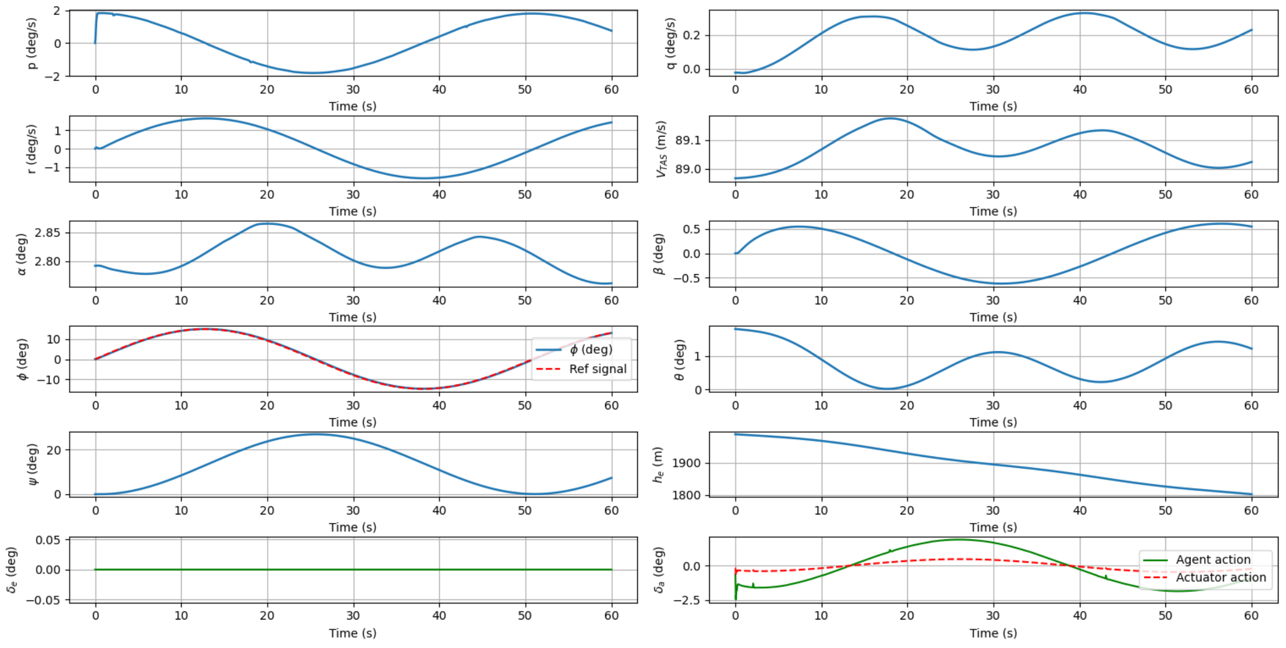


Figure 7.3: RED5Q quarter actuator efficiency condition roll control behaviour

7.3 Actuator jolt condition behaviour

The pitch control behaviour of the actuator jolt condition is given in Figure 7.4. The agent is able to recover from large disturbances and return to the given pitch reference. Furthermore, the agent acts as expected with the elevator being pushed in the opposite direction of the disturbance. However, something unexpected happens starting at 10 seconds, when the maximally negative disturbance is given, where the aircraft model does not seem to be as responsive compared to the maximally positive disturbance at 5 seconds. The reason behind the asymmetry in pitch responsiveness to maximal elevator deflection is not fully understood since the agent is able to achieve high pitch angles under nominal conditions.

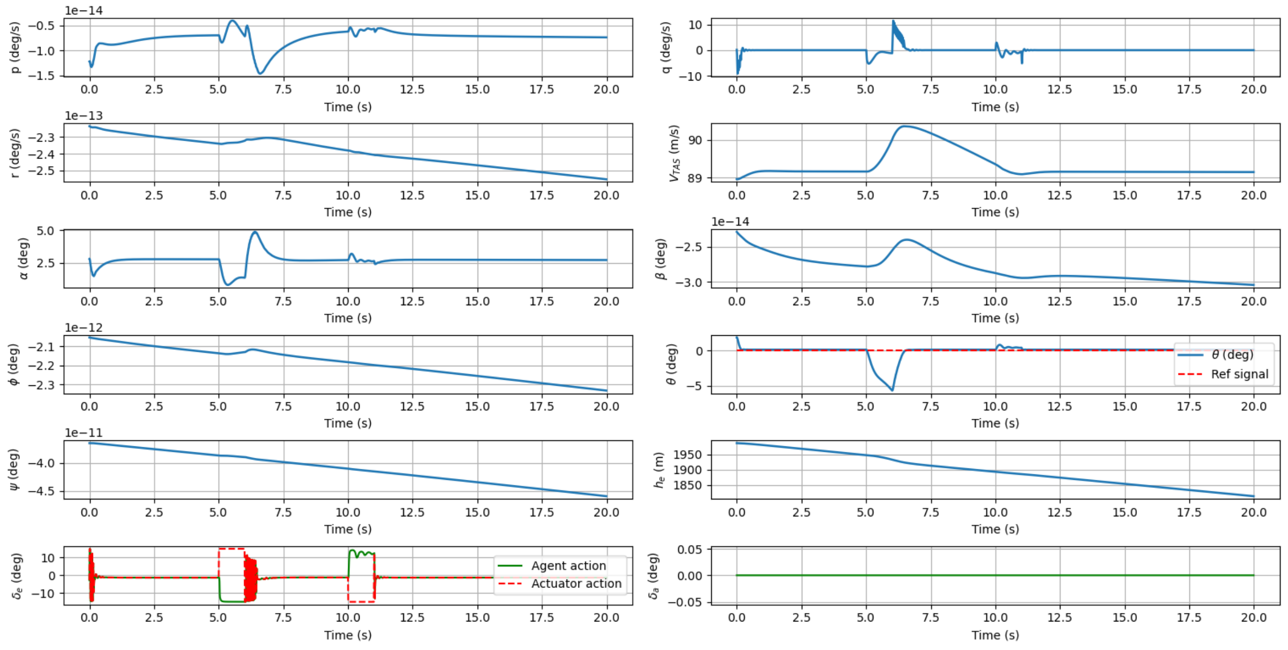


Figure 7.4: SAC actuator jolt condition pitch control behaviour

The biaxial control behavior under the jolt condition is shown in Figure 7.5. Once again, the agent successfully recovers from disturbances on both the pitch and roll axes. The previously observed unexpected asymmetric responsiveness to maximal longitudinal disturbances is absent, which adds to the mystery of why it occurs only during pitch control disturbances. Overall, the response of REDQ agents to the jolt condition is not always great as discussed in the article. Additionally, it also not always smooth. This is because the large deflection forces agents to non-nominal states within the environment, where it likely had little training. Smoothness issues are also likely related to the fact that the larger the reference error, the larger the agent's actions are expected to be to compensate for the reference error.

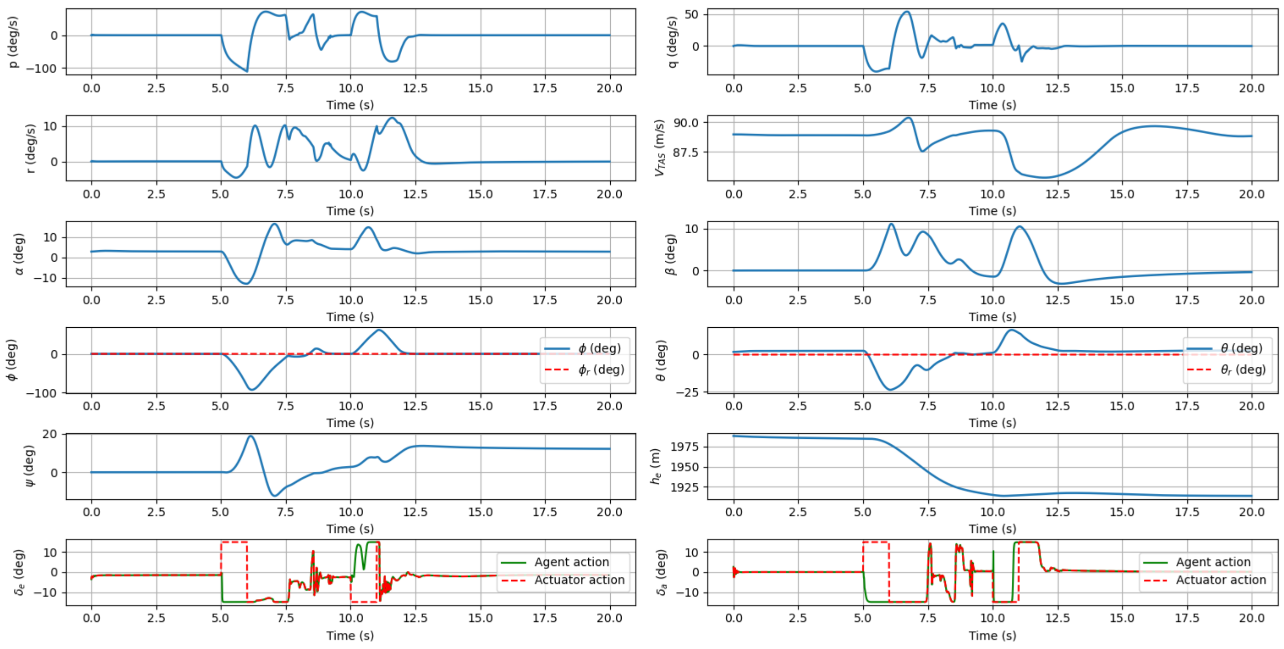


Figure 7.5: RED3Q actuator jolt condition biaxial control behaviour

Chapter 8

Verification and Validation

This chapter discusses the verification procedures followed throughout the research done in this thesis. Section 8.1 covers the verification performed, which consisted of a model match check to verify the compilation of the MATLAB CitAST model was done successfully and an RL implementation verification which revolved around sanity checks and analysing agent policies. Validation of the model was not conducted in this research, as the model has previously been validated by van den Hoek et al. [34] where real flight data was used to identify and validate the CitAST model for the PH-LAB.

8.1 Verification

The verification of procedures throughout the research can be divided into model verification and RL implementation verification. These are addressed in Subsection 8.1.1 and Subsection 8.1.2 respectively.

8.1.1 Verifying the model

To verify that the PH-LAB MATLAB model had been compiled to the Python extension module correctly, the trimmed and zero input response of the model was recorded to compare if it matched. The model was compiled with the default values that it came with (different from the operating point used for the rest of the thesis) just to ensure that there were no errors in the compiling. For brevity, only the zero input response is given in Figure 8.1 for the compiled Python extension module and Figure 8.2 gives the original MATLAB PH-LAB model response. It is worthwhile to note that the aircraft crashes near 180 seconds, which explains the sudden jumps in the pitch rate, pitch angle and angle of attack.

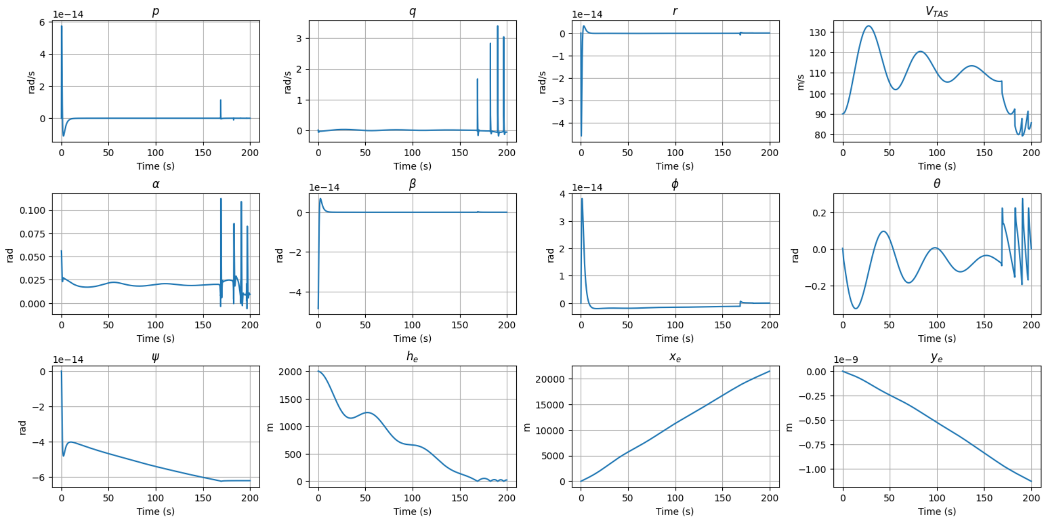


Figure 8.1: Zero input response compiled python extension module

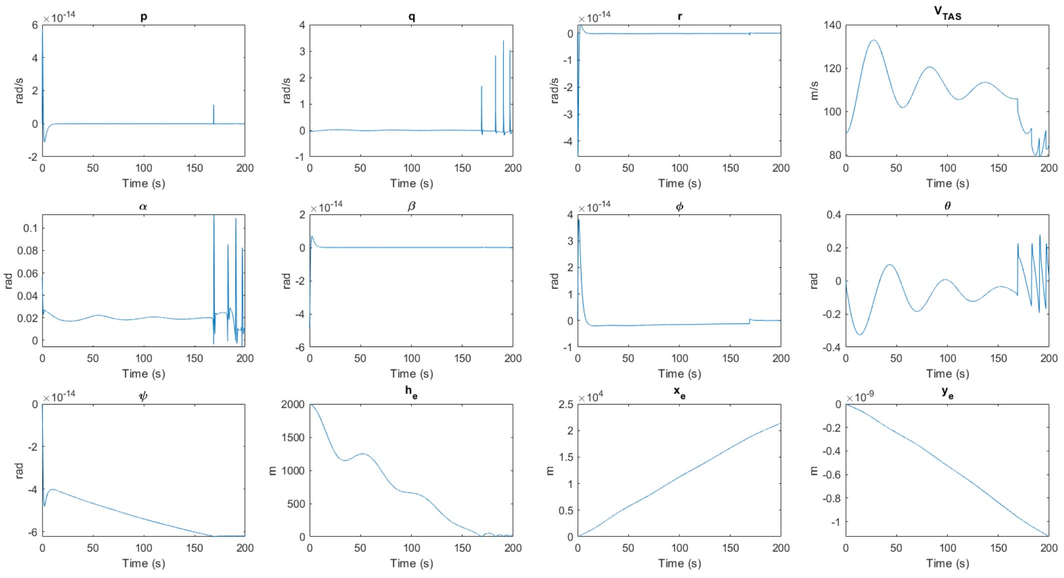


Figure 8.2: Zero input response MATLAB PH-LAB model

8.1.2 Verifying RL implementation

While developing the RL implementation, several sanity checks were performed to ensure the learning was going as expected. Firstly, the implementations of the SAC and REDQ agents were forked from those used for the preliminary analysis which were shown to have comparable behaviour with the SB SAC implementation. The agent implementations were also able to successfully learn policies in the hinged and inverted pendulum environments. Secondly, the learning behaviour of the agents for the attitude tracking tasks was actively monitored by generating plots to check their tracking performance, loss curves and episodic returns. These plots were used to make sure that the agents were learning as intended. Lastly, the agent policies were plotted to make sure these were consistent with the expected behaviour. This last procedure is explained at length hereafter, it is useful to keep in mind the sign conventions defined in Figure 8.3.

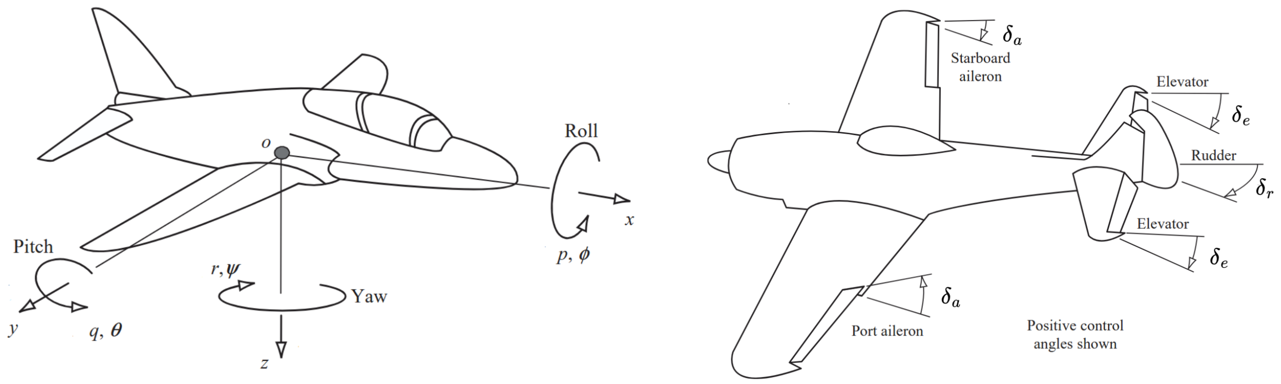


Figure 8.3: Relevant aircraft states and actuators sign convention adapted from Cook [53]

The policy plot for the pitch control agent can be seen in Figure 8.4. The leftmost plot was constructed using only the deterministic policy actions (so no sampling from a distribution was done). The middle plot was constructed with the stochastic policy actions, which explains the more noisy and less smooth behaviour shown. The rightmost plot is a scatter plot showing the trajectory spread during learning where the states were logged every step. Note that the shape of the plots can change from agent to agent as there can be multiple policies that can be learned to be good enough from the training.

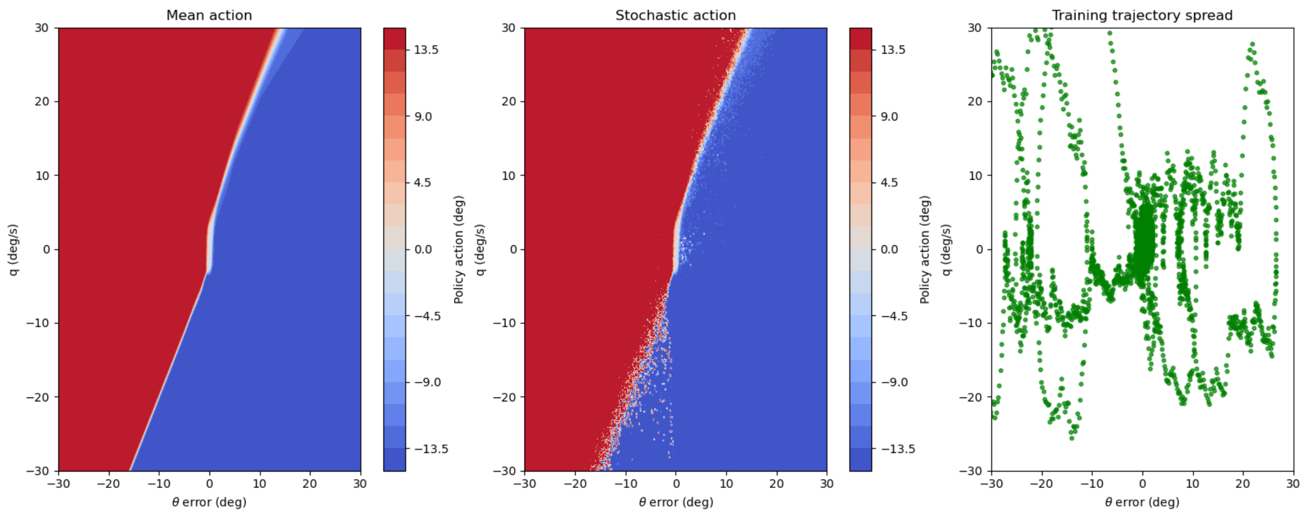


Figure 8.4: Pitch control SAC agent policy plot

To interpret this plot, it is useful to remember that $\theta_e = \theta_{ref} - \theta$ and consult the sign convention seen in Figure 8.3 as for the pitch control agent the action corresponds to the elevator deflection. Looking at when the error is zero and the pitch rate is very positive, it makes sense for the elevator deflection to be maximally positive in order to stabilise the incoming pitch angle increase. Likewise, when the error is zero and the pitch rate is very negative, one sees a maximally negative elevator deflection. Looking at when the pitch rate is zero and the error is large $\theta_e = \theta_{ref} - \theta$ is maximised when θ is extremely negative, meaning the aircraft is pitching down and so a negative action (nose up) is stabilising. The same line of argument is used to explain why when the pitch rate is zero and the error is very negative, you get a positive action. The diagonal lines are expected due to the aircraft's C_{m_α} being negative, so when the angle of attack is changed due to pitch changes there is a restorative moment that resists the change. The reason why the diagonal decision boundaries are not in a straight line is not fully known, but the angle of the lines is highly sensitive to the seed of the training environment. It is therefore believed to be a combination of what the agent was exposed to during the training, as well as the nonlinear nature of lift curves.

A zoomed in version of the plots can be seen in Figure 8.5 where the more relevant agent actions are visible as in nominal conditions it will be close to the reference signal so the error will be small.

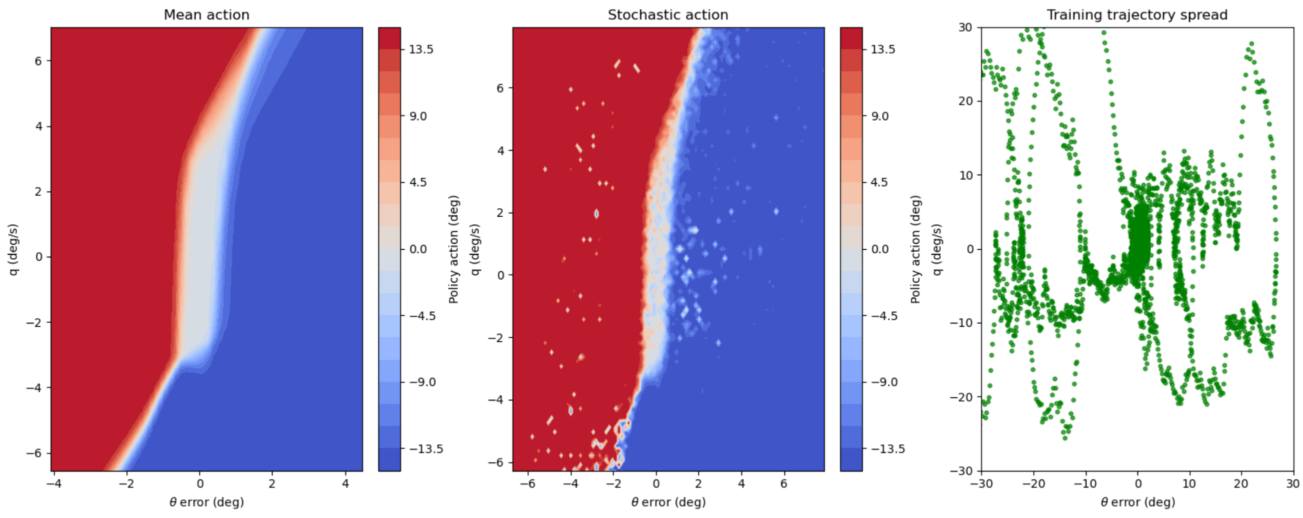


Figure 8.5: Zoomed in pitch control SAC agent policy plot

A policy plot for the roll control agents can be seen in Figure 8.6. The structure is the same as that for pitch except that now the error is defined in ϕ instead of θ and the action now corresponds to the aileron deflections.

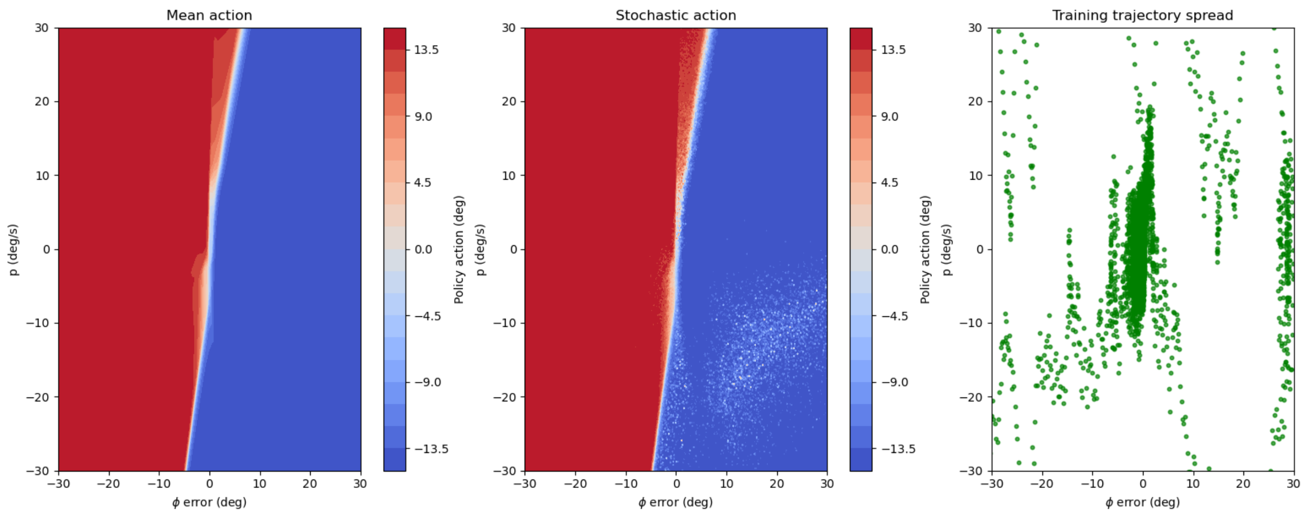


Figure 8.6: Roll control SAC agent policy plot

It is once again useful to remember the reference and sign convention seen in Figure 8.3 together with the roll error definition $\phi_e = \theta_{ref} - \phi$. As can be seen, when the error is zero and the roll rate is very positive, the action is very positive as this causes a negative roll moment. Similarly, when the error is zero and the roll rate is very negative, a negative action is chosen to create a positive roll moment. When the roll rate is zero and the roll error is very negative, this means the aircraft is at a very positive roll rate (follows from roll error definition) and the agent choses a positive action to counteract this. The same logic applies when the error is very positive, except that the agent now chooses a very negative action to counteract the very negative roll angle. The fact that the decision boundary is slightly skewed is in this time due to the negative restorative moment C_{l_p} caused partly by the asymmetry in the lift generated by the wings when the aircraft rolls (as for the rest the aircraft is symmetrical with a small dihedral angle of 4.5° [54]).

A more zoomed in version of the plots can be seen in Figure 8.7 where the more relevant parts of the policy is shown under nominal conditions.

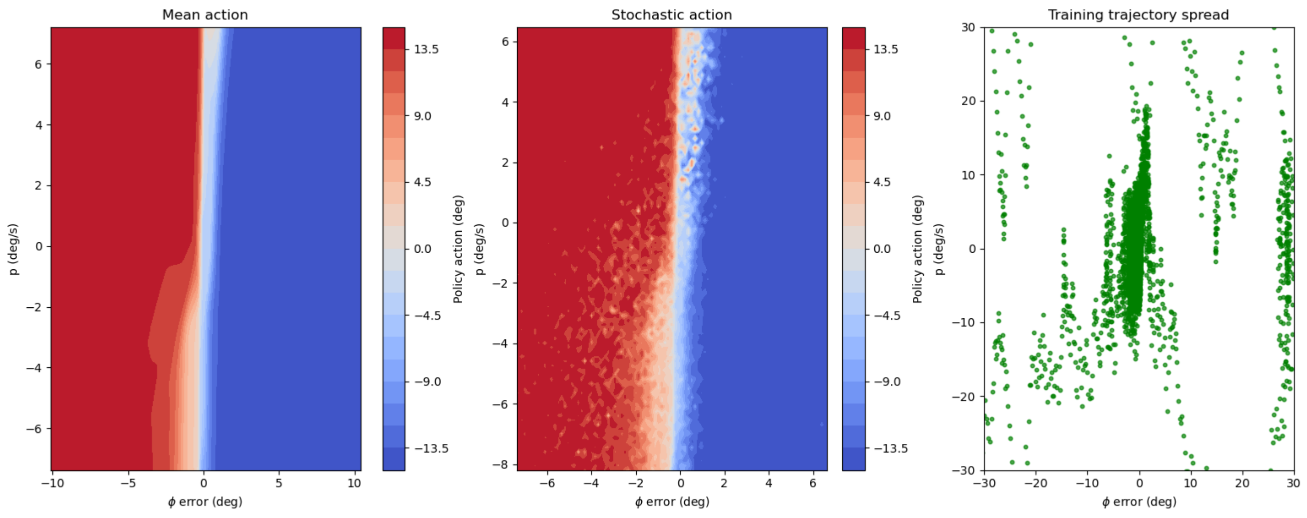


Figure 8.7: Zoomed roll control SAC agent policy plot

Figure 8.8 shows the policy plot for the biaxial control task generated by fixing the states given at the top of the plot to zero. The policies generated are consistent with those discussed previously. The one visible difference is that the aileron policy is not as vertically divided. However, in the nominal performance region where the error and roll rate are low it remains consistent.

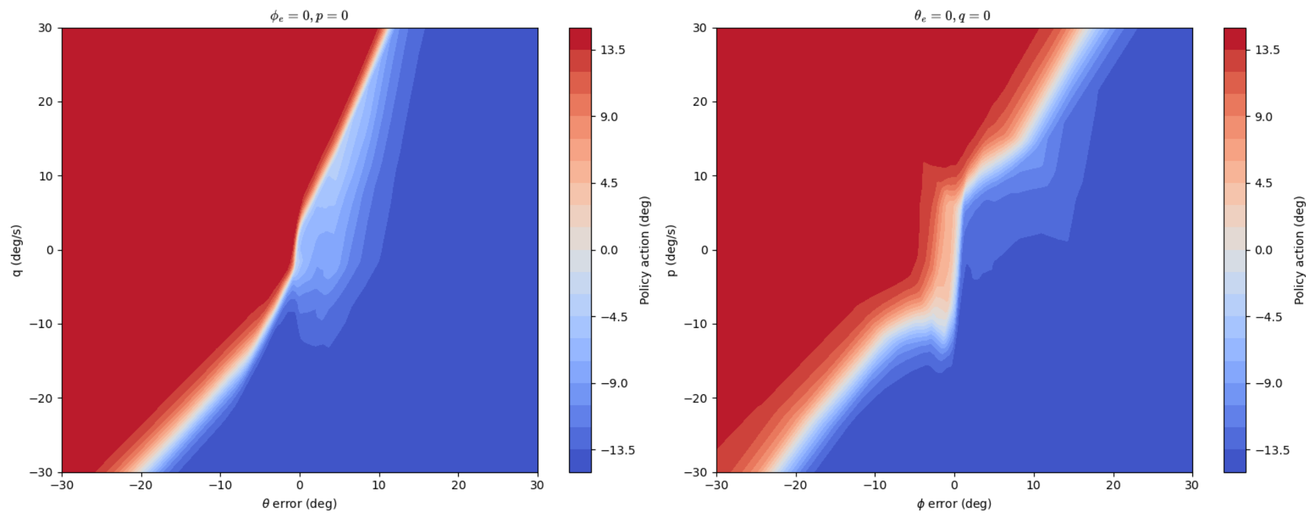


Figure 8.8: Biaxial control task policy plot for SAC agent, elevator action on left and aileron on right

Part IV

Closure

Chapter 9

Conclusion

Reinforcement learning offers an alternative to PID controllers for flight control applications. The main benefits of a model free RL solution is that they require no knowledge of plant dynamics and can bypass the need for gain scheduling used in conventional PID control techniques. This is achieved by agents being able to generalise beyond operating points. Sample efficiency is central to any machine learning approach as it can constrain which techniques are feasible. The research in this thesis adds to the body of knowledge of how RL based model free flight control can be made more sample-efficient in offline training while also illustrating the possible robustness consequences. The following discussion will detail how the research questions posed were answered and the research objective of improving the sample efficiency of model free flight control reinforcement learning approaches was achieved.

Q1 How can reinforcement learning with a focus on sample efficiency be used for flight control applications?

1. How can RL be used to solve control problems?
2. What are some relevant performance metrics to keep in mind when developing RL for flight control purposes?
3. What is a suitable methodology to tackle sample efficiency improvement?

These questions will be addressed in the order they appear. Firstly, RL can be used to solve control problems formulated as MDPs. In the field of flight control, several applications were mentioned such as the flying of a quadrotor drone with a PPO algorithm and using an IDHP agent to learn a fault tolerant pitch control task online. This is covered in Chapter 2. Secondly, there are several relevant metrics to keep in mind when developing a RL solution in flight control among which are safety, sample efficiency and the explainability of the solution. Abstract mathematical definitions are presented as the precise definitions are problem dependent. The specifics of these definitions are found in Section 2.6. To answer the last question, the sample efficiency was tackled by using the REDQ algorithm on flight control tasks. REDQ increases sample efficiency by generalising the number of critics in an actor critic architectures, choosing a subset size over which to minimise during updates and increases the number of updates performed per batch update step. Further details on REDQ are given in Section 3.2

Q2 How can a preliminary analysis investigating sample efficiency be implemented on benchmark environments?

1. What learning environments will be used to test the methodology?
2. What baseline algorithms will be used in the preliminary analysis for comparison?
3. How will the sample efficiency of the preliminary analysis be computed?

The questions will now be addressed in order of appearance. Firstly, the preliminary analysis was implemented in two environments the "Pendulum-v1" and "InvertedPendulum-v4" from the Gymnasium and Mujoco libraries by interfacing with their environments. These two were chosen because one of them was simple to learn for the agents which was good for testing and verification purposes while the other posed a more complex task. More detail on these environments is given Section 4.1. Secondly, the baseline algorithm used during the preliminary for comparison was SAC and it was chosen because it already had good base performance and REDQ had hyperparameters specific to the SAC implementation for reference. Section 4.2 gives an overview of the procedure performed. Lastly, learning curves and environment specific sample efficiency metrics were defined in order to be able to compare the REDQ algorithm's performance relative to the baseline SAC. Section 4.2 also discusses this.

Q3 How does the proposed methodology perform on the Cessna Citation PH-LAB Model?

1. How can the RL algorithm be interfaced with the PH-LAB model?
2. What algorithms will be used as comparisons in the methodology?
3. How will the proposed methodology improve sample efficiency?
4. What are the safety implications for the proposed methodology?

These questions will be discussed in the order presented. All of the information regarding these research questions was covered in the scientific article in Part I which is used as reference to substantiate the claims mentioned hereforth. Firstly, RL algorithms were interfaced with a Python compatible compiled version of the PH-LAB model by framing the problem as an MDP. This involved using the state space of the PH-LAB model to formulate an agent, its actions, a reward function and the plant dynamics as the environment. Secondly, the comparison algorithm used in the methodology was once again SAC. However, it was implemented on three attitude tracking agents for pitch, roll and biaxial control. Using the SAC agents performance as a benchmark, a comparison was made between standard SAC performance and that of REDQ for all three tasks. Thirdly, the proposed methodology improved sample efficiency by showing the learning curves of SAC and REDQ agents in conjunction with a statistical analysis to demonstrate that the gains in sample efficiency during initial training were statistically significant. Threshold performance metrics were also defined to get a sense of how long it took the agents on average to achieve these thresholds. Lastly, the safety implications in the methodology were asymptotical learning instability, robustness in fault scenarios and actuator concerns on real systems from volatile command signals. REDQ agents were shown to have worsened asymptotical learning stability in the learning curves. To test the robustness of REDQ agents, a fault tolerance analysis was conducted which demonstrated diminished robustness in REDQ agents. This was visible in the number of outliers during fault tests, with performance being the worst in fault tests that introduced an actuator jolt that pushed them far from nominal conditions. The drawbacks in learning stability and robustness were attributed to overfitting. Extra information regarding the fault tolerance analysis and how concerns related to the action volatility of the agents were addressed see Chapter 7 and Section 6.1 respectively. Early stopping and active visualisation of actuator outputs during training are recommended as preventative measures to mitigate safety concerns.

To conclude, this research demonstrated the viability of using RL as an alternative to conventional PID control. The RL agents were able to successfully control the model without anything resembling gain scheduling and were even able to generalise under fault scenarios, albeit with varying levels of robustness success. For the three attitude control tasks performed in the experiment, the average accurate tracking convergence (error $< 1^\circ$) occurred within 5,500 training steps for pitch, 6,400 for roll and 11,500 for biaxial control. For all tasks, REDQ demonstrated faster initial learning which fulfilled the learning objective of improving the sample efficiency of model free flight control reinforcement learning approaches.

Chapter 10

Recommendations

During the thesis process, several ideas and directions were considered but not pursued due to time and focus considerations. This chapter will discuss a selection of them.

1. **Implementing other REDQ compatible algorithms**

   The thesis mainly focused on SAC and it's REDQ counterpart. However, the ideas behind REDQ are applicable to all actor critic architectures. Implementing REDQ across a family of algorithms would give a broader look at how the sample efficiency of different algorithms can be improved for flight control. Given how quickly the agents were able to learn the control tracking tasks, it could be the case that more deterministic and safe exploration-centric algorithms would be able to learn the task online (through clever use of techniques like a warm start). Two immediate candidates that come to mind would be the online-offline architectures developed in Teirlinck [55] and dos Santos [56].

2. **Full attitude tracking controller**

   The agents developed during this research were limited to attitude control within two axes, namely pitch and roll. This was because the yaw damper that came implemented with the model hindered the learning in the yaw axis. To investigate full attitude tracking performance the yaw damper could be removed from the MATLAB model to test if the same methodology would yield good performance. Given the ease with which the agent was able to generalise to more than uni axial attitude control, the methodology outlined during this thesis has a strong case for working for full attitude control, whether it's all done by one agent or each axis done by separate agents. Regardless, full attitude control was not evaluated in this paper so it remains a recommendation.

3. **Further robustness testing** Robustness testing through fault tolerance analysis was limited to three cases due to time and technological considerations, compiling the CitAST model into Python is an elaborate process. A step to improve on this would be to test the agents behaviours across a larger suite of tests. Immediate candidates for this would be to test the agents behaviour starting from different trimmed conditions than it was trained for or untrimmed conditions altogether. For additional reference, Dally [37] analysed the robustness of their SAC controller with additional cases such as a jammed rudder, saturated elevator range, icing and centre of gravity shift.

4. **Investigate sensitivity to REDQ and CAPS hyperparameters**

   The research in this paper looked at two REDQ agents: RED3Q and RED5Q. As such, the research only presented results with matching number of critics and update to data ratio. Though an additional hyperparameter optimisation study on REDQ was conducted, its objective metric was long term mean performance as opposed to sample efficiency. A more granular sensitivity analysis could be performed to investigate how the sample efficiency performance of the agent varies with respect to individual changes in the number of critics or the update to data ratio changes while keeping everything else equal. Additionally, the learning related effects of CAPS hyperparameters was not investigated.

5. **Policy transfers and policy transfer learning**

   The policies learned by agents on the PH-LAB model could be transferable to other models. The smallest incremental step take in this direction would be to take the trained agents and assess their performance on models of aircraft with similar aerodynamic characteristics and dynamics. Even if the performance is not ideal, the trained neural network could still serve as a starting point for learning the task either through further training of the agent network or by using the policy to fill a replay buffer to help train a new off-policy agent. For such use cases, training agents using sparse reward functions should be investigated as potential better reward shaping candidates due to their more abstract nature.

Bibliography

[1] International Air Transport Association (IATA). Global outlook for air transport: A world with lower oil prices? Technical report, International Air Transport Association, December 2024. URL `https://www.iata.org/en/iata-repository/publications/economic-reports/global-outlook-for-air-transport-december-2024/`.

[2] Ramesh Konatala, Erik-Jan Van Kampen, and Gertjan Looye. Reinforcement learning based online adaptive flight control for the cessna citation ii (ph-lab) aircraft. In *AIAA Scitech 2021 Forum*, page 0883, 2021.

[3] Chao Wang, Jian Wang, Yuan Shen, and Xudong Zhang. Autonomous navigation of uavs in large-scale complex environments: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 68(3):2124–2136, 2019. doi: 10.1109/TVT.2018.2890773.

[4] Pouria Razzaghi, Amin Tabrizian, Wei Guo, Shulu Chen, Abenezer Taye, Ellis Thompson, Alexis Bregeon, Ali Baheri, and Peng Wei. A survey on reinforcement learning in aviation applications. *Engineering Applications of Artificial Intelligence*, 136:108911, October 2024. ISSN 0952-1976. doi: 10.1016/j.engappai.2024.108911. URL `http://dx.doi.org/10.1016/j.engappai.2024.108911`.

[5] X. Chen, C. Wang, Z. Zhou, and K. Ross. Randomized ensembled double q-learning: Learning fast without a model. 2021. URL `https://www.scopus.com/inward/record.uri?eid=2-s2.0-85150281117&partnerID=40&md5=0194d3d5c7b87619a22fa6799ebf4d3c`.

[6] M. Tim Jones. Models for machine learning. `https://developer.ibm.com/articles/cc-models-machine-learning/`, 2017. Accessed: 2024.

[7] Yuankun Liu, Dongxia Zhang, and Hoay Beng Gooi. Optimization strategy based on deep reinforcement learning for home energy management. *CSEE Journal of Power and Energy Systems*, 6(3):572–582, 2020. doi: 10.17775/CSEEJPES.2019.02890.

[8] David Silver, Julian Schrittwieser, Karen Simonyan, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. doi: 10.1038/nature24270. URL `https://doi.org/10.1038/nature24270`.

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

[10] Matteo Hessel Diana Borsa, Hado van Hesselt. Deepmind x ucl rl lecture series. Lecture - reinforcement learning, UCL & Google DeepMind, 2021.

[11] Vlad Gavra. Evolutionary reinforcement learning: Hybrid approach for safety-informed fault-tolerant flight control. Master's thesis, Delft University of Technology, May 2024.

[12] Ligang Gong, Qing Wang, Changhua Hu, and Chen Liu. Switching control of morphing aircraft based on q-learning. *Chinese Journal of Aeronautics*, 33(2):672–687, 2020. ISSN 1000-9361. doi: https://doi.org/10.1016/j.cja.2019.10.005. URL `https://www.sciencedirect.com/science/article/pii/S1000936119304066`.

[13] Amir Ramezani Dooraki and Deok-Jin Lee. An innovative bio-inspired flight controller for quad-rotor drones: Quad-rotor drone learning to fly using reinforcement learning. *Robotics and Autonomous Systems*, 135:103671, 2021. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2020.103671. URL `https://www.sciencedirect.com/science/article/pii/S092188902030511X`.

[14] Sean R. Sinclair, Siddhartha Banerjee, and Christina Lee Yu. Adaptive discretization in online reinforcement learning, 2022. URL `https://arxiv.org/abs/2110.15843`.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL `https://arxiv.org/abs/1312.5602`.

[16] Yaakov Engel, Shie Mannor, and Ron Meir. Bayes meets bellman: The gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 154–161, 2003.

[17] Idan Achituve, Aviv Shamsian, Aviv Navon, Gal Chechik, and Ethan Fetaya. Personalized federated learning with gaussian processes, 2021. URL `https://arxiv.org/abs/2106.15482`.

[18] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[19] Nathan O Lambert, Daniel S Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer SJ Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

[20] Nursultan Imanberdiyev, Changhong Fu, Erdal Kayacan, and I-Ming Chen. Autonomous navigation of uav by using real-time model-based reinforcement learning. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1–6, 2016. doi: 10.1109/ICARCV.2016.7838739.

[21] Qingyan Huang. Model-based or model-free, a review of approaches in reinforcement learning. In *2020 International Conference on Computing and Data Science (CDS)*, pages 219–221, 2020. doi: 10.1109/CDS49703.2020.00051.

[22] Sinan Çalışır and Meltem Kurt Pehlivanoğlu. Model-free reinforcement learning algorithms: A survey. In *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4, 2019. doi: 10.1109/SIU.2019.8806389.

[23] Biao Luo, Derong Liu, Tingwen Huang, and Ding Wang. Model-free optimal tracking control via critic-only q-learning. *IEEE Transactions on Neural Networks and Learning Systems*, 27(10):2134–2144, 2016. doi: 10.1109/TNNLS.2016.2585520.

[24] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[25] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[26] Steven C. H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey, 2018. URL `https://arxiv.org/abs/1802.02871`.

[27] Stefan Heyer, Dave Kroezen, and Erik-Jan Van Kampen. Online adaptive incremental reinforcement learning flight control for a cs-25 class aircraft. In *AIAA Scitech 2020 Forum*, page 1844, 2020.

[28] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020. URL `https://arxiv.org/abs/2005.01643`.

[29] Haoran ZHAO, Hang FU, Fan YANG, Che QU, and Yaoming ZHOU. Data-driven offline reinforcement learning approach for quadrotor's motion and path planning. *Chinese Journal of Aeronautics*, 2024. ISSN 1000-9361. doi: https://doi.org/10.1016/j.cja.2024.07.012. URL `https://www.sciencedirect.com/science/article/pii/S100093612400267X`.

[30] Rafael Figueiredo Prudencio, Marcos R. O. A. Maximo, and Esther Luna Colombini. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Transactions on Neural Networks and Learning Systems*, 35(8):10237–10257, August 2024. ISSN 2162-2388. doi: 10.1109/tnnls.2023.3250269. URL `http://dx.doi.org/10.1109/TNNLS.2023.3250269`.

[31] Farama Foundation. Mujoco environments. `https://gymnasium.farama.org/main/environments/mujoco/`, 2024. Accessed: 2024-08-08.

[32] Gabriel Dulac-Arnold, Daniel J. Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *CoRR*, abs/1904.12901, 2019. URL `http://arxiv.org/abs/1904.12901`.

[33] Aviv Tamar, Yonatan Glassner, and Shie Mannor. Optimizing the cvar via sampling, 2014. URL `https://arxiv.org/abs/1404.3862`.

[34] M. A. van den Hoek, C. C. de Visser, and D. M. Pool. Identification of a cessna citation ii model based on flight test data. In *Advances in Aerospace Guidance, Navigation and Control*, pages 259–277. Springer International Publishing, 2018.

[35] Klaas-Jan van Woerkom. Nlr, tui fly en lufthansa maken proefvluchten op twente airport. Online, sep 2021.

[36] Wing Chan. Sample-efficient reinforcement learning for flight control. Master's thesis, Delft University of Technology, August 2024.

[37] Killian Dally. Deep reinforcement learning for flight control. *AIAA 2022-2078*, 2021. doi: 10.2514/6.2022-2078.

[38] Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning, 2021. URL `https://arxiv.org/abs/2007.04938`.

[39] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015. URL `https://arxiv.org/abs/1509.06461`.

[40] 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, 2019. URL `https://arxiv.org/abs/1509.02971`.

[41] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL `https://arxiv.org/abs/1801.01290`.

[42] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018. URL `https://arxiv.org/abs/1802.09477`.

[43] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018. URL `https://arxiv.org/abs/1802.09477`.

[44] Tim Miller. Policy gradients: Rl notes, 2023. URL `https://gibberblot.github.io/rl-notes/single-agent/policy-gradients.html`. Accessed: 2024-09-10.

[45] Farama Foundation. Gymnasium pendulum environment documentation, 2023. URL `https://gymnasium.farama.org/environments/classic_control/pendulum/`.

[46] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2019. URL `https://arxiv.org/abs/1709.06560`.

[47] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019. URL `https://arxiv.org/abs/1812.05905`.

[48] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[49] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24, page 4. Curran Associates, Inc., 2011. URL `https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf`.

[50] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 754–762, Bejing, China, 22–24 Jun 2014. PMLR. URL `https://proceedings.mlr.press/v32/hutter14.html`.

[51] Siddharth Mysore, Bassel Mabsout, Renato Mancuso, and Kate Saenko. Regularizing action policies for smooth control with reinforcement learning, 2021. URL `https://arxiv.org/abs/2012.06644`.

[52] Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning, 2022. URL `https://arxiv.org/abs/2110.02034`.

[53] Michael V. Cook. *Flight Dynamics Principles.* Elsevier Ltd., 2013.

[54] Laurens van Horssen, Coen De Visser, and Daan Pool. Aerodynamic stall and buffet modeling for the cessna citation ii based on flight test data. 01 2018. doi: 10.2514/6.2018-1167.

[55] Casper Teirlinck. Reinforcement learning for flight control. Master's thesis, Delft University of Technology, September 2022.

[56] Lucas dos Santos. Safe & intelligent control. Master's thesis, Delft University of Technology, July 2023.

[57] Stephen H. Lane and Robert F. Stengel. Flight control design using non-linear inverse dynamics. *Automatica*, 24(4):471–483, July 1988. ISSN 00051098. doi: 10.1016/0005-1098(88)90092-1. URL `https://linkinghub.elsevier.com/retrieve/pii/0005109888900921`.

[58] Tarek Madani and Abdelaziz Benallegue. Backstepping Control for a Quadrotor Helicopter. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3255–3260, Beijing, China, October 2006. IEEE. ISBN 978-1-4244-0258-8 978-1-4244-0259-5. doi: 10.1109/IROS.2006.282433. URL `http://ieeexplore.ieee.org/document/4058900/`.

[59] Hyo-Sung Ahn, YangQuan Chen, and Kevin L. Moore. Iterative Learning Control: Brief Survey and Categorization. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 37(6):1099–1121, November 2007. ISSN 1094-6977. doi: 10.1109/TSMCC.2007.905759. URL `http://ieeexplore.ieee.org/document/4343981/`.

[60] Stefan Heyer. Reinforcement Learning for Flight Control: Learning to Fly the PH-LAB. Master's thesis, Delft University of Technology, March 2019.

Appendix A

Planning

Thesis

The planning for the thesis was broken into four main phases, the work packages associated with each phase are denoted by Work Packages (WP) and their scheduling is shown in Gantt Chart - A with week zero being 21/11/2024 and week 28 being 03/07/2025. Once the draft has been submitted and feedback has been received, the final thesis will be handed in on 25/06/2025 and defended on 09/06/2025. These cover the four main goals of this literature study:

1. Setup an experiment for flight control tasks
2. Evaluate agent learning for different control tasks
3. Answer research questions
4. Write a scientific article
5. Defend the thesis

WP 1: Thesis Reporting

This spans the whole thesis as documenting the progress, uploading results and managing referencing is a continuous task.

WP 1.1: Report writing & formatting

This package encompasses the continuous reporting, referencing and formatting.

WP 1.2: Documenting results

The documentation of results is a recurring task that is repeated every time relevant results are generated which are to be used in the thesis.

WP 2: Pitch Control Task

This part of the thesis encapsulates the period that was spent developing just the pitch tracking controller. This first task is expected to take the longest as a lot of time is spent onboarding onto a non-familiar coding environment.

WP 2.1: Set up PH-LAB model

Setting up the PH-LAB will involve compiling a MATLAB model into a Python extension module. This process is not straight forward; it involves some knowledge of the MATLAB compiler and how to use SWIG to generate a Python extensions out of compiled C code.

WP 2.2: Interface model with agents

Interfacing the PH-LAB model with agents will require creating code around the python extension module to: interface the actions the agent outputs as inputs to the model and use the outputs of the model for subsequent states and reward allocation.

WP 2.3: Hyperparameter tuning

This part of the thesis will involve searching for a set of hyperparameters that allow the agent to learn the tasks. Will most likely involve an Optuna study similar to what was done during the preliminary analysis.

WP 2.4: Verification & validation

Verification and validation is something that will be done in every task. For the pitch control task, it will involve ensuring the code is working as intended through sanity checks based on outputs and visual feedback from plots.

WP 3: Roll Control Task

This part of the thesis contains all the packages related to the development of the roll tracking controllers.

WP 3.1: Expand pitch control code for roll

It is expected for a lot of the code for the roll tracking task to be very similar to the code of the pitch control task. With some changes on which inputs and outputs are used from the PH-LAB model, so the code needs to be expanded for this functionality.

WP 3.2: Verification & validation

Verification will consist of similar measures as described for the pitch control task. A comparison of the pitch and roll control tasks can also be done at this stage since the pitch controller will already be finished to make sure they're learning comparably.

WP 4: Biaxial Control Task

The biaxial control task will be a controller that controls both the pitch and the roll. It will be the final results related task in the thesis.

WP 4.1: Generalise code for biaxial control

The code must be expanded for compatibility of agents with a larger state and action space. Functionalities like the training, evaluation and interfacing of the agents will need to be changed as well. The plotting functionalities will also require changes.

WP 4.2: Define experiment evaluation criteria

With functioning code for all tasks, the experiment evaluation criteria will be implemented. This will involve a sample efficiency and fault tolerance analysis. The scope of the analysis will be fixed at the end of this task.

WP 4.3: Verification & validation

This package encompasses the verification that will be done for the biaxial controller. It will use similar verification checks as those performed for pitch and roll.

WP 4.4: Generating final results

the final plots and performance metrics will be generated at this stage for final inclusion in the thesis.

WP 5: Defence Preparation

The defence is the final phase of the thesis where the work will be defended in front of a committee.

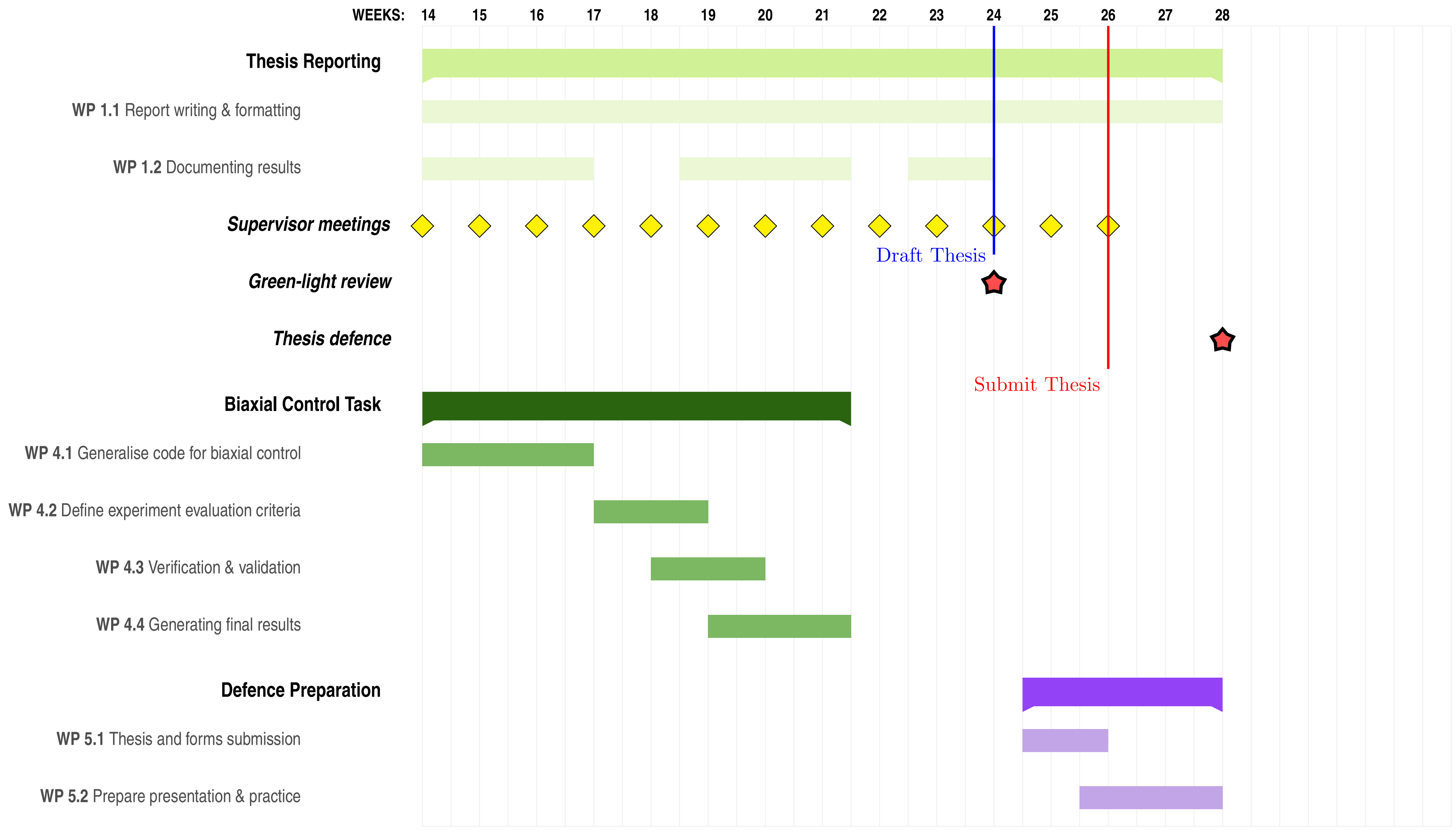
WP 5.1: Thesis and forms submission

This package contains the procedural tasks that are done near the end of the thesis.

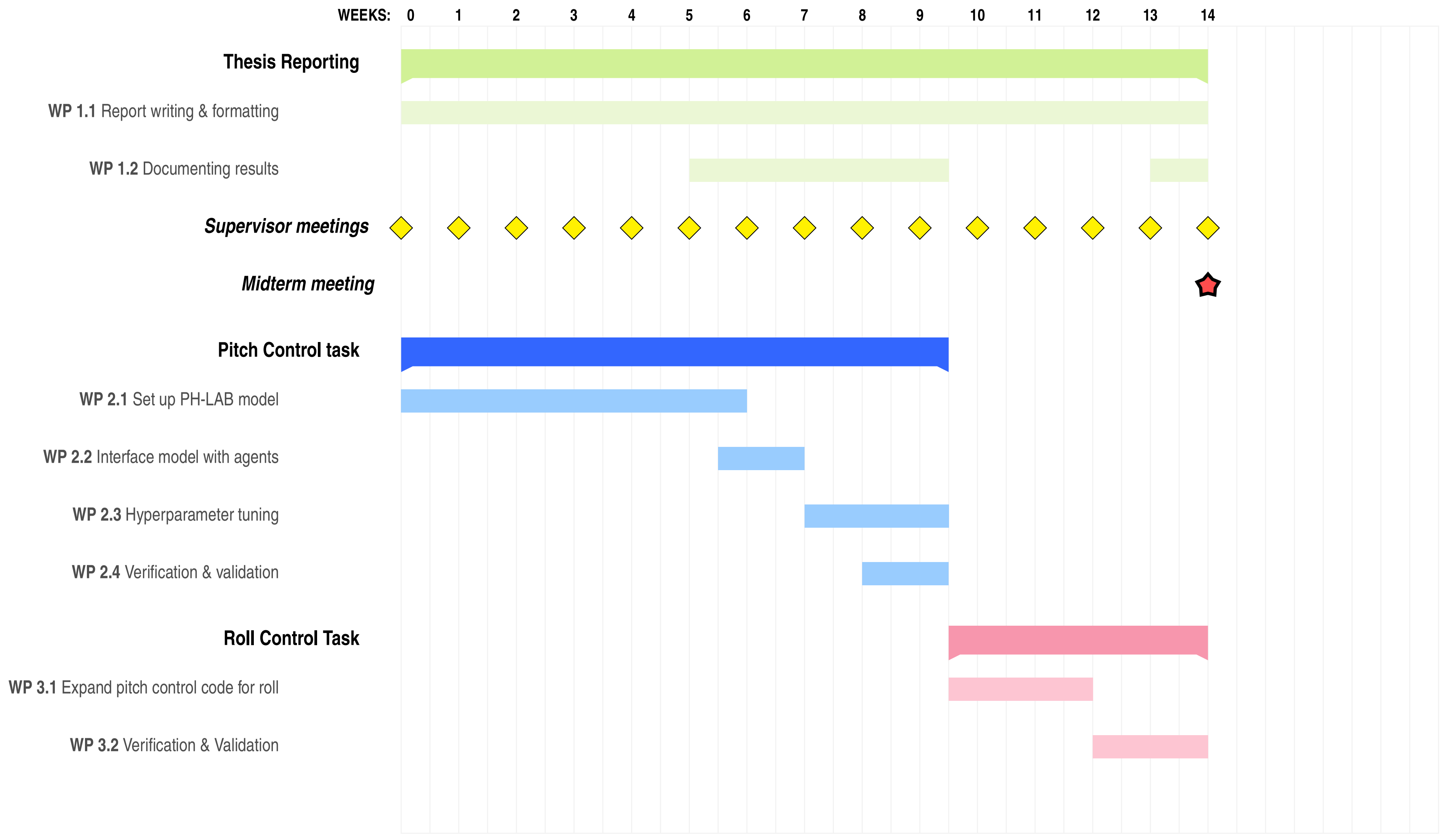
WP 5.2: Prepare presentation & practice

This will be the final task where the presentation to be given during the thesis will be made. It will require practising and studying the content for any incoming questions.

A. Gantt Chart - Thesis



A. Gantt Chart - Thesis



Literature study

The planning for the literature study was broken into four main phases, the work packages associated with each phase are denoted by Work Packages (WP) and their scheduling is shown in Gantt Chart - B with week zero being 26/06/2024 and week 21 being 20/11/2024. Once the draft has been submitted and feedback has been received, the final literature study will be handed in at some date TBD. These cover the three main goals of this literature study:

1. Brush up on the knowledge acquired throughout academic years
2. Explore research done in order to formulate research gaps/questions & solution methods/approaches
3. Perform a preliminary analysis in a benchmark environment using the proposed methods to assess their feasibility prior to the thesis assignment

WP 1: Planning & Reporting

This part of the literature study spans the whole literature study as documenting the knowledge acquired and managing referencing is a continuous task.

WP 1.1: Report writing & formatting

This package encompasses the continuous reporting, referencing and formatting.

WP 1.2: Planning Lit. Study

This package encompasses the planning that is scheduled outright from the beginning of the literature study. It could be the case that some tweaks to the planning are performed throughout the whole literature study but foundation is done in the first four weeks.

WP 2: Refresher - RL & Flight Control

This work phase was planned for only four weeks and acts as a baseline which will aid in digesting the upcoming literature review.

WP 2.1: Reinforcement learning recap

The refresher on reinforcement learning will cover the theory behind it, the taxonomies & categories associated with it as well as touch on some variations of it which are particularly suited for flight control.

WP 2.2: Control problems in aerospace

The refresher on control problems in aerospace will briefly introduce the type of control problems that arise in aerospace applications and with an emphasis on those particularly suited to be solved via reinforcement learning.

WP 3: Literature Review

The aim of the literature review phase is to dive into what's being done or has been done in order to get a feel of where the cutting edge of research is currently at in order to identify a research gap/question(s) and form an idea of how to design a methodology that will allow for the research question(s) to be answered.

WP 3.1: Identify research gap & question(s)

During this part of the literature review, the focus will be in researching possible knowledge gaps and formulating questions. This will be done in parallel with WP 3.2 as researching state of the art will aid in finding out exactly where the gap is as well as which questions to ask. This is where the scope of the literature study is fixed.

WP 3.2: Researching state of the art

In parallel with WP 3.1, this package will focus will be in investigating what the cutting edge research is for reinforcement learning algorithms and how it's being used for flight control purposes.

WP 3.3: Define thesis approach/methods

With the research gap and question(s) formulated, what follows is coming up with methods that will successfully answer the research question(s). This might be by coming up with a RL algorithm, improving on one that's been established or applying it to an area of application which has not yet been tested.

WP 4: Benchmarking

This phase of the literature study will act as a way of comparatively testing (relative to other established algorithms) whether the proposed methodology appears promising, the aim is to reduce the amount of future iterations/re-scoping by evaluating early on if the chosen research question & methodology that will result in a fruitful thesis assignment.

WP 4.1: Familiarize with Mujoco benchmark

The Mujoco benchmarks are collections of environments used to test RL algorithms. The idea is to familiarize with how to set it up by creating a python environment with all necessary dependencies and libraries.

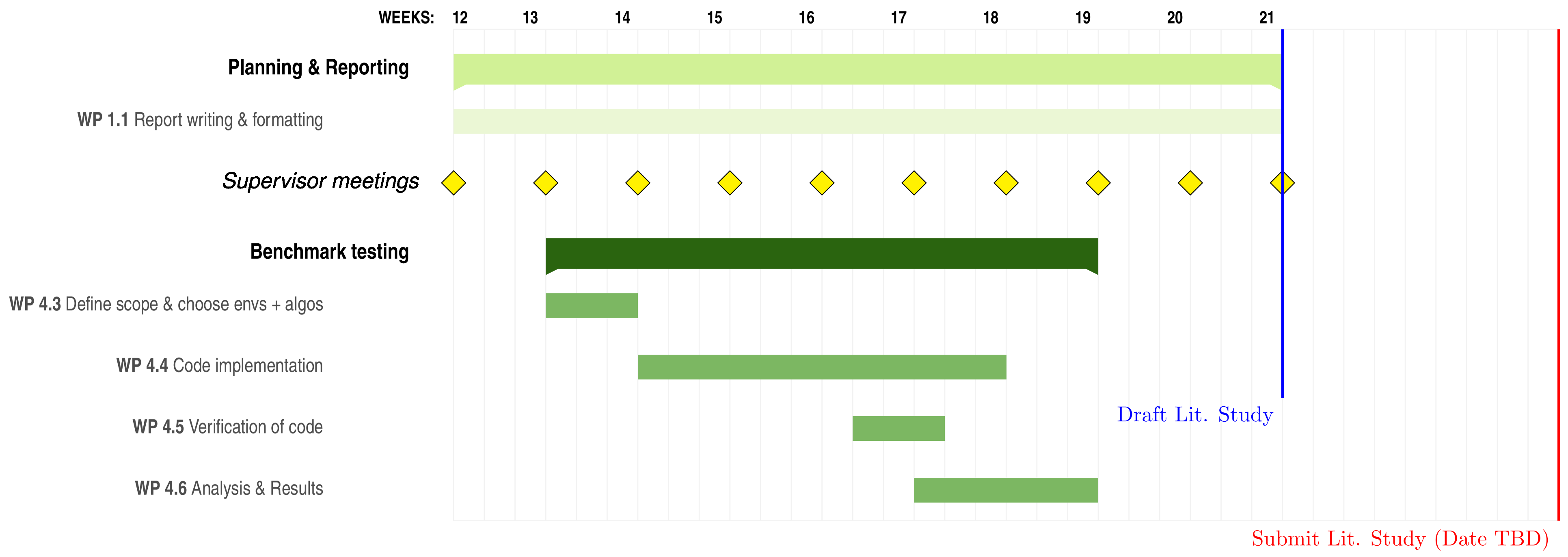
WP 4.2: Try out different algs & envs

During this period different environments/algorithms will be run and measured in the Mujoco environments (most likely in the inverted pendulum). The relevant metrics that are used to measure performance will be explored and understood.

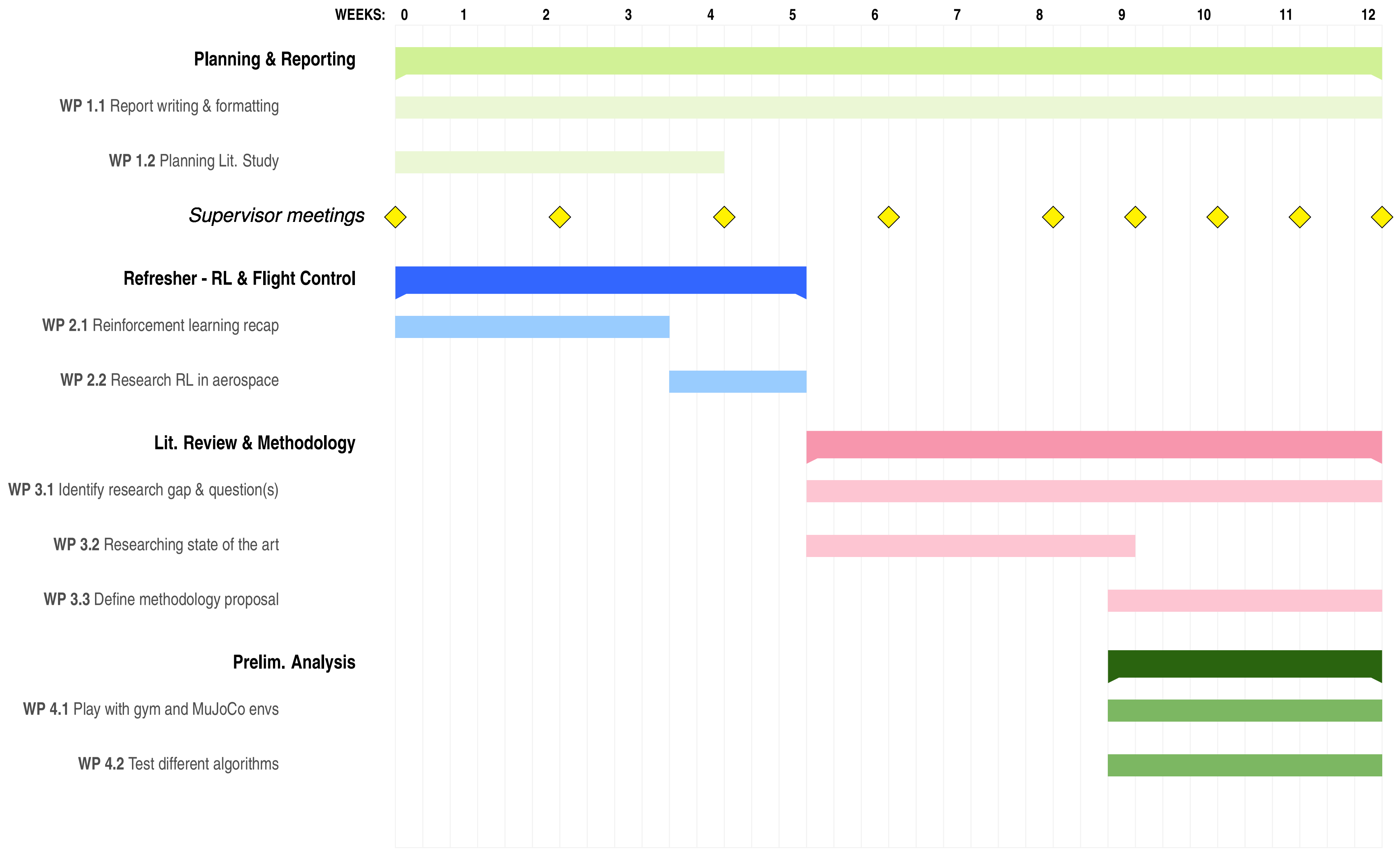
WP 4.3: Implement possible algorithms

In this last phase of the benchmarking, the proposed methods will be implemented to see their feasibility and determine if they look promising enough to extend it to the flight control problems in the scope of the research.

B. Gantt Chart - Literature Study



B. Gantt Chart - Literature Study


WEEKS: 0 1 2 3 4 5 6 7 8 9 10 11 12
Planning & Reporting
WP 1.1 Report writing & formatting
WP 1.2 Planning Lit. Study
Supervisor meetings
Refresher - RL & Flight Control
WP 2.1 Reinforcement learning recap
WP 2.2 Research RL in aerospace
Lit. Review & Methodology
WP 3.1 Identify research gap & question(s)
WP 3.2 Researching state of the art
WP 3.3 Define methodology proposal
Prelim. Analysis
WP 4.1 Play with gym and MuJoCo envs
WP 4.2 Test different algorithms