# Evolutionary Reinforcement Learning

## A Hybrid Approach for Safety-informed Intelligent Fault-tolerant Flight Control

Vlad Gavra

TU Delft

# Evolutionary Reinforcement Learning

A Hybrid Approach for Safety-informed Intelligent Fault-tolerant Flight Control

Thesis report

by

Vlad Gavra

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on January 19, 2023 at 14:00

| *Thesis committee*: | |
|---|---|
| Chair: | Dr. ir. M.M. van Paassen, |
| Supervisor: | Dr. ir. E. van Kampen |
| External examiner: | Dr. S.J. Hulshoff |
| Additional examiner: | ir. I. El-Hajj |
| Place: | Faculty of Aerospace Engineering, Delft |
| Project Duration: | February 2022 - January 2023 |
| Student number: | 4663632 |

Faculty of Aerospace Engineering · Delft University of Technology

# Preface

Artificial Intelligence is already revolutionising the world. With this thesis, I was able to combine my curiosity for AI and my genuine fascination for things that fly. In my research, I try to synchronise the latest advancement in bio-inspired intelligent methods, such as deep reinforcement learning and evolutionary algorithms, to develop adaptive controllers that are robust to faults, disturbances and unexpected phenomena. I am confident that my work will contribute towards safer autonomous systems and I hope to, at some point in future, see them flying.

Continuing the effort of previous students and researchers in this field, the code developed as part of this thesis has been made publicly available[1].

By submitting this thesis, I am concluding a five-year journey of my life. Working to become an Aerospace Engineer at TU Delft has inevitably shaped the way I learn and think. It has been a truly unforgettable experience for which I shall thank many inspiring people that I have met along the way. Among all of them, I would like to thank Erik-Jan van Kampen for his much-needed guidance throughout my nine-month research. Your ability to always show a calm approach towards any seemingly unsolvable problem inspired me. Also, by always making sure that I am spending my resources exploiting the most important aspects, you led me to achieve what I now feel has been the perfect balance between learning new topics and delivering relevant results.

This journey would not have been possible without the support of my friends made in Delft and Oradea, alongside whom I learnt so much. We have shared both adventures and stressful work sessions, highs and lows, and then debated all of them in endless discussions. I will always cherish these memories.

Thank you Casi for your continuous support, trust and encouragement. Without these, I would have lost my confidence many times and I am more than grateful that you have never let that happen. Last but not least, I would particularly like to thank my parents for devoting their entire attention and effort to support me in achieving my goals.

This is all thanks to you. Mulțumesc!

[1]`https://github.com/VladGavra98/SERL`

# Contents

# List of Figures

# List of Tables

| Acronyms | Definition |
|---|---|
| ABS | Adaptive Backstepping |
| ADP | Approximate Dynamic Programming |
| AI | Artificial Intelligence |
| ANN | (Artificial Neural Network |
| BC | Behaviour Characteristic |
| BO | Bayesian Optimisation |
| BP | Back Propagation |
| CG | Centre of Gravity |
| CMA | Covariance Matrix Adaptation |
| CMA-ME | Covariance Matrix Adaptation for Multi-dimensional Archive of Phenotypic Elites |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| DL | Deep Learning |
| DDPG | Deep Deterministic Policy Gradient |
| DOF | Degree of Freedom |
| DP | Dynamic Programming |
| DQN | Deep Q-Network |
| DRL | Deep Reinforcement Learning |
| EA | Evolutionary Algorithm |
| ELU | Exponential Linear Unit |
| EOM | Equations of Motion |
| ES | Evolutionary Strategies |
| ERL | Evolutionary Reinforcement Learning |
| FCS | Flight Control System |
| FTC | Fault-tolerant Control |
| GA | Genetic Algorithm |
| GPI | Generalized Policy Iteration |
| GPU | Graphics Processing Unit |
| iADP | Incremental Dynamic Programming |
| IBS | Incremental Backstepping |
| IFCS | Intelligent Flight Control System |
| IID | Independent and Identically Distributed |
| INDI | Incremental Non-linear Dynamic Inversion |
| ISA | International Standard Atmosphere |
| LOC | Loss of Control |
| MC | Monte Carlo |
| ML | Machine Learning |
| eVTOL | Electrical Vertical Take-off and Landing |
| MAP-Elite | Multi-dimensional Archive of Phenotypic Elites |
| MDP | Markov Decision Process |
| MSBE | Mean Squared Bellman Error |
| MSE | Mean Squared Error |
| NEAT | Neuro-Evolution of Augmenting Topologies |
| NN | Neural Network |

| Acronyms | Definition |
|---|---|
| NS | Novelty Search |
| OBM | On-board Model |
| PDERL | Proximal Distilled Evolutionary Reinforcement Learning |
| PID | Proportional Integral Derivative |
| POMDP | Partially Observable Markov Decision Process |
| QD | Quality-Diversity |
| ReLU | Rectified Linear Unit |
| RL | Reinforcement Learning |
| RMSE | Root Mean Square Error |
| RNN | Recurrent Neural Network |
| SAC | Soft Actor-Critic |
| SGD | Stochastic Gradient Descent |
| TD | Temporal Difference |
| TD3 | Twin Delayed Deep Deterministic Policy Gradient |
| TPU | Tensor Processing Unit |
| UAS | Unmanned Aerial System |
| UAV | Unmanned Aerial Vehicle |

## Mathematical and Physical Symbols

| Symbol | Definition | Unit |
|---|---|---|
| $a$ | Agent action | [-] |
| $b$ | Bias vector | [-] |
| $B$ | Batch size | [-] |
| $\mathbf{b}$ | Behaviour characteristic array | [-] |
| $F$ | Fitness function value | [-] |
| $l$ | Sample loss value | [-] |
| $\mathcal{L}$ | Loss function | [-] |
| $n$ | Total number of steps/iterations | [-] |
| $N$ | Population size | [-] |
| $\mathcal{P}$ | Probability distribution function | [-] |
| $Q$ | State-action value function | [-] |
| $r, \tilde{r}$ | Reward value | [-] |
| $R$ | Return | [-] |
| $s$ | Agent state | [-] |
| $Sm$ | Smoothness of the action signal | $[rad \cdot Hz]$ |
| $S_{zz}$ | Spectral density function of signal $z$ | [[z]/$Hz$] |
| $T$ | Thrust value | [-] |
| $V$ | State value function | [-] |
| $W$ | Weights matrix | [-] |
| $x_t, \dot{x}_t$ | Horizontal position and velocity of Lander | $[m, m/s]$ |
| $y_t, \dot{y}_t$ | Vertical position and velocity of Lander | $[m, m/s]$ |
| $\alpha_{\text{ac./cr.}}$ | Learning rate used for the actor/critic learning | [-] |
| $\gamma$ | Discount factor of the future rewards | [-] |

| Symbol | Definition | Unit |
|---|---|---|
| $\delta$ | Temporal difference error | [-] |
| $\epsilon$ | Random variable sampled from a Gaussian probability distribution, innovation (Chapter 8 only) | [-] |
| $\eta$ | Entropy temperature | [-] |
| $\kappa$ | Genetic buffer size, forgetting factor (Chapter 8 only) | [-] |
| $\lambda$ | Regularisation coefficient | [-] |
| $\phi$ | Value/critic parameters | [-] |
| $\theta$ | Actor/policy parameters | [-] |
| $\theta_t$, $\dot{\theta}_t$ | Orientation angle and velocity of the LunarLander | $[rad,\, rad/s]$ |
| $\mu$ | Deterministic actor (policy) function | [-] |
| $\pi$ | Stochastic actor (policy) function | [-] |
| $\sigma$ | Standard deviation | [-] |
| $\rho$ | Polyak coefficient | [-] |
| $p, q, r$ | Roll, pitch, yaw rate | $[rad/s]$ |
| **x** | Dynamic state vector | [-] |
| **u** | Control input vector | [-] |
| $q_\infty$ | Free-flow dynamic pressure | $[N/m^2]$ |
| $V_{tas}$ | True Air-speed | $[m/s]$ |
| $\alpha, \beta$ | Angle of attack, sideslip | $[rad]$ |
| $\delta_e, \delta_a, \delta_r$ | Elevator, aileron, rudder deflection | $[rad]$ |
| $\phi, \theta, \psi$ | Roll, pitch, yaw angle | $[rad]$ |

## Sets and Tuples

| Symbol | Definition |
|---|---|
| $\mathcal{A}$ | Set of all valid actions |
| $B$ | Batch of previous experiences |
| $\mathcal{B}$ | Space spanned by attainable behaviours |
| $\mathcal{D}$ | Buffer of previous experiences |
| $\mathcal{D}_x$ | Domain to which $x$ belongs |
| $\mathcal{S}$ | Set of all valid states |
| $T_k$ | Trajectory of the $k$ previously reached states |
| $\mathcal{T}_t$ | Transition tuple at time $t$ |

## Subscripts and Superscripts

| Symbol | Definition |
|---|---|
| $x_t$ | Time step |
| $x^*$ | Optimal function value |
| $x^{(i)}$ | Data sample index |
| $x_k$ | Iteration index |
| $x_C$ | Crossover |
| $x_M$ | Mutation |

# 1 Introduction

The aviation sector has experienced an excellent safety record in recent decades, becoming the safest means of long-distance transportation. Continuing an ever-lowering rate, the overall number of aviation crashes decreased by $36\%$ between 2018 and 2019 [1].

Despite this, the number of accidents involving ‘loss of control in flight’ is not decreasing at a comparable rate. Improving the autonomy capabilities of aerospace systems in general, and of aircraft controllers in particular, could cut the number of casualties even further. Arguably, autonomy shall strive to become a reliable safety net acting at multiple levels of the aerospace industry.

Breakthroughs in intelligent control systems allow for increased autonomy and better human-machine teaming. There is hope that modern techniques incorporating bio-inspired Artificial Intelligence (AI) could enhance the safety, durability, autonomy, and performance of aerospace vehicles. Ultimately, a milestone in achieving these goals concerns improving the way simulation-based flight control systems are able to remain robust to real-life conditions such as faults, unmodeled dynamics and unforeseen disturbances [2, 3, 4].

## Deep Reinforcement Learning

Reinforcement Learning (RL) is a framework that uses repeated rewarded interactions between an intelligent agent and its environment to learn what it should do to perform the task better [5].

Historically, RL has first been applied to optimal flight control as Approximate Dynamic Programming (ADP). This framework relies on online identification of a dynamics model and demonstrated robustness in dealing with faults [6]. Also, it proved successful in controlling highly complex coupled systems such as the F-16 fighter aircraft [7] due to its higher approximation power than other non-linear control methods such as Incremental Non-linear Dynamic Inversion (INDI) or incremental backstepping (IBS) [2, 8]. Furthermore, it proved capable of learning to control autonomous agents that can fly and perform tasks such as learning acrobatic flight from human demonstration [9].

Recently, research in offline methods aims to improve performance by training longer offline in a simulation environment. Harnessing deep neural networks as function approximators [10], Deep Reinforcement Learning (DRL) has obtained mainstream popularity due to its ability to reach above-human-level performance in complex tasks such as playing arcade games from images, driving autonomous cars or wining against world-class players in strategy games [11]. Within the category of offline methods, the state-of-the-art frameworks are currently mainly based on the following four algorithms: DDPG [12], TD3 [13], SAC [14] and PPO [15].

Moreover, the DRL framework inherits the advances developed in the field of deep learning (DL). Essentially, deep neural networks provide a relatively-sample efficient way to train highly non-linear functions to approximate complex dynamics [5, 10]. Leveraging longer training times, DRL obtains state-of-the-art performance in tasks considered difficult for more traditional control systems. These include reference tracking in navigation [16, 17] and low-level attitude control in presence of faults [18] or disturbances [19] without any prior knowledge of the environment or solution.

The offline model-free DRL methods prioritise performance, often learning policies that command actions with high levels of noise. By doing so, they aggravate the gap between simulation and reality, increasing the challenge of deployment on flight hardware [20]. Regularisation methods have been developed to inhibit the action noise when controlling multi-rotor vehicles [20]. The root cause of the noise has yet to be extensively discussed.

In terms of the performed optimisation, DRL generically belongs to the class gradient-based methods as the neural networks are trained by updating their parameters by stepping the gradient values back-propagated from a loss function [10].

## Neuro-evolutionary Algorithms

Biological evolution has been creating complex organisms capable of sustaining life in their own unique way for billions of years [21]. This process is simulated by evolutionary algorithms. These gradient-free methods iteratively optimise a set of control strategies, or population, by selecting the fittest (i.e., highest scoring) individuals in each generation [22, 23]. Variation operators such as mutation and cross-over are applied in each generation to explore new candidates or exploit already optimised solutions. Similar to their use in DRL, the control policies can be approximated by neural networks. This is known as neural evolution, an alternative to gradient-based optimisation [10, 24].

The Neuro-Evolution of Augmenting Topologies (NEAT) algorithm was one of the field's early successes, employing a novel graph-based technique to efficiently encode the values of neural network weights [25, 26]. The same technique allowed neural networks to evolve with increasing size, essentially optimising policies with minimal complexity [24]. This was especially effective in early applications to RL tasks [27].

Within this category, a new paradigm referred to as *novelty search*, has emerged with the goal to search for a set of individuals, maximally diverse with respect to the space of attainable behaviours [28, 29]. Branching from it, the quality-diversity method has become a compromise between the search for both fitness and novelty [30]. It learns a maximally diverse repertoire of high-quality policies which are stored in a map-like memory archive spanned by the resulting behaviours. The intelligent agent can then select the best-performing control policy for its current task and condition either by estimating the performance using a pre-defined model of the environment dynamics [31] or by learning online a locally-accurate approximation of the world model [32, 33].

Evolutionary Algorithms have been proven successful when optimising highly non-convex functions with sparse gradients by escaping local optima or saddle points [34, 35]. Moreover, they always produce a locally optimised repertoire of behaviours mapped by control policies (e.g.robots walking in every direction) that are relatively close to the global optimum [21]. This has already been applied to adaptive robots [21], closing the gap between the ideal world of simulations and reality with unexpected conditions, unpredictable or unmodelled phenomena, faults and failures [31, 36].

## Hybrid Methods

For a long time, people have realised that the extraordinary adaptability of living species stems from the effect of the self-development and learning mechanism of an individual combined with evolutionary processes acting at the population level [37, 38]. Whiteson et.al. combine NEAT with an RL algorithm to achieve sample-efficient reinforcement learning [39], whereas in [40] the authors use neural evolution to improve exploration in stochastic domains. Last but not least, EAs have been used to optimise the hyperparameters of DRL algorithms, in the field known as AutoRL [41].

Despite their evident achievements, both the aforementioned bio-inspired classes have their own drawbacks [37]. Researchers had the idea of directly combining both frameworks in learning algorithms to combat each other's disadvantages [37, 42]. Hybrid algorithms incorporating evolutionary loops and reinforcement learning updates are a novel class of solution showing both higher sample efficiency and enhanced performance in complex environments while benefiting from stable learning [43, 44].

## Research Gap

In spite of the success of DRL algorithms in flight control and the demonstrated ability of population-based methods to remain fault-tolerant and change-resistant, none of the hybrid ERL methods has been applied to a flight control task or tested on faulty systems. The current research aims to fill this gap.

Section 1.1 discusses the research questions formulated to help bridge the gap, as well as the objective associated with them. Then, Section 1.2 defines the scope in which these questions will be answered.

## 1.1. Research Design

The present work aims to fulfil the following objective:

**Research Objective**

Improve robustness of fault-tolerant intelligent flight control by adapting a hybrid evolutionary reinforcement learning framework to provide reference-tracking control for a high-fidelity model of a fixed-wing aircraft.

As a means to reach the above goal, research questions and sub-questions have been identified. The main research questions of the present work are split into two distinct blocks, namely “Method Feasibility” and “Implementation and Robustness”.

**Method Feasibility**

**Met-1** What are the requirements for intelligent fault-tolerant flight control?
- **Met-1.a** What are the most critical faults a modern flight control system shall adapt to?
- **Met-1.b** What are the limitations of intelligent flight controllers in dealing with system faults?

**Met-2** What are the state-of-the-art offline hybrid methods combining deep reinforcement learning and evolutionary algorithms?
- **Met-2.a** What are the strengths and limitations of deep reinforcement learning?
- **Met-2.b** What are the strengths and limitations of evolutionary algorithms?
- **Met-2.c** Which hybrid methods are proposed in the field’s literature?
- **Met-2.d** What benchmarks and corresponding metrics could be used to compare their control performance?

**Met-3** Which offline-trained hybrid framework is feasible to maximise the chosen metric for the control of a high-dimensional continuous system in the presence of faults?

The first question, **Met-1**, dives into the field of fault tolerance in aerospace control. As an answer to its first sub-question, a set of fault cases is compiled by reviewing both industrial standards and the field literature. Then, sub-question **Met-1.b** looks at how other intelligent control algorithms had already been applied to fault-tolerant aerospace control tasks. By answering both sub-questions, the need for a fault-tolerant controller and the identified limitations of current approaches will be translated into qualitative requirements in terms of performance, data efficiency and robustness. The method currently developed shall aim to pass those requirements.

Then, the focus moves to the bio-inspired frameworks used for intelligent control. More specifically, the literature is reviewed to identify the state-of-the-art offline intelligent control methods that combine Deep Reinforcement Learning and Evolutionary Algorithms. This is completed by first researching the strengths and limitations of each framework as answers to sub-questions **Met-2.a** and **Met-2.b**. Then, sub-question **Met-2.c** is answered by reviewing the available hybrid methods and sub-question **Met-2.d** by identifying a way to test the performance of such methods and benchmark their performance.

Within the first block, the final question relates the found intelligent control architectures to the task of controlling a faulty system. Seeking an answer to question **Met-3** implies showing that at least one hybrid controller can be trained in simulation on a nominal task and robustly control the same system in presence of faults. then, this method becomes the baseline algorithm to be built upon.

After the feasibility of a novel hybrid control method is shown, the focus shifts towards implementing and improving its robustness for a relevant aerospace application. This process follows the second set of research questions:

**Implementation and Robustness**

**Imp-1** What is a suitable system/platform to train and test the hybrid flight controller?
- **Imp-1.a** What simplified model and which initial conditions can be used to train the controller offline?
- **Imp-1.b** What should be the architecture of the control loop, observation space and action space?
- **Imp-1.c** Which fault conditions should be evaluated during testing?

**Imp-2** How can an offline learnt set of policies be efficiently and safely updated for online control of the selected platform in presence of faults?
- **Imp-2.a** What are the most promising ways to use safety information to improve average controller performance with respect to the baseline hybrid framework?
- **Imp-2.b** What is a sample-efficient method to select a well-behaving policy during online control of the faulty system?

**Imp-3** How does the hybrid approach compare to an intelligent control framework based on DRL in terms of performance under faulty conditions?

Question **Imp-1** deals with the system side of the control task. Whereas the answer to sub-question **Imp-1.a** looks toward a simplified aircraft model and trim conditions, the second sub-question is concerned with the architecture of the control loop and the state-action space. Finally, sub-question **Imp-1.c** seeks to identify which fault cases from the answer to question **Met-1** are to be modelled as test cases for the developed controllers.

Next, the research investigates how the hybrid controller can be improved and used in a sample-efficient manner. These challenges are directly addressed by the research question **Imp-2**, with its first sub-question focusing on the possible improvements. Then, a key aspect to be researched is the possibility to select which policy, among the evolved population, will control the system. This raises the sub-question **Imp-2.b**.

Last but not least, the hybrid framework will be compared to an intelligent controller trained by a state-of-the-art DRL technique. To make them comparable, the two frameworks will be trained using the same settings and tested against the same task of controlling the system in presence of the aforementioned faults and conditions, tied to the answer to sub-question **Imp-1.c**. The comparative analysis is meant to answer **Imp-3**, the final research question.

## 1.2. Scope of Research

The control tasks considered only concern offline continuous control. Starting from the preliminary analysis, the selected algorithms are trained and tested in a simulated environment. On one hand, testing on real hardware poses additional difficulties such as sensor noise, environmental disturbances or other unpredictable dynamics. On the other hand, a flight testing campaign is many orders of magnitude more expensive and time-consuming than running simulations [3].

The preliminary analysis considers a simplified optimal control problem. Then, moving to the second set of research questions, the rest of the results are related to the task of aircraft attitude control. Here, the simulation environment for learning and evaluation considers continuous attitude control of a fixed-wing aircraft model. The model with high-fidelity non-linear plant dynamics is compiled as a standard Flight CAD package referred to as DASMAT [45]. It was designed by [45] within the Control & Simulation department of the TU Delft Faculty of Aerospace Engineering to model the Cessna 500 aircraft. Later, it has been improved by [46] through system identification on flight test data recorded on-board the PH-LAB, the research aircraft jointly owned between the TU Delft Faculty of Aerospace Engineering and Netherlands Aerospace Centre (NLR) [1].

## 1.3. Inter-field Terminology

In the next chapters, work from the fields of deep reinforcement learning, optimal control and neuro-evolutionary algorithms will be cited, reviewed and implemented. On multiple occasions, the same aspects

[1]https://cs.lr.tudelft.nl/citation/

might be referred to with different terminologies, depending on the method's field of origin. Table 1.1 has been compiled to limit the amount of confusion and more easily distinguish between the key terminologies commonly used within each field.

**Table 1.1:** Differences in terminology and notation between the control, RL and EA communities.

| **Control Theory** | **Deep Reinforcement Learning** | **Neuro-evolutionary Algorithms** |
|---|---|---|
| plant/system | agent | evolutionary agent |
| control law | policy/actor | individual |
| x,u | s,a | *both* |
| cost-to-go | value function | n.a. |
| objective, cost | return,loss | fitness, - |
| iteration | trial/episode | generation |
| model parameters/coefficients | weights & biases | genes (genome) |
| persistent excitation | exploration | variation operators |

Notably, in control theory, one can either try to maximise an objective function (optimal control frameworks) or minimise a cost or cost-to-go value. The EA approaches traditionally only consider the fitness of individuals - a metric they aim to maximise.

## 1.4. Report Structure

The report consists of four parts. The first one, Part I, contains the Scientific Article which presents the theoretical background, summarises the followed methodology and discusses the main results of this thesis. Then, the second part covers the literature study and the preliminary analysis, which will review, identify and verify a hybrid learning method for the task of fault-tolerant flight control.

Part III moves on and showcases additional results of the experimental studies: hyperparameter tuning in Chapter 5, the impact of the safety-informed mutation operator in Chapter 6 and the aircraft's simulated time traces in the presence of faults and unseen flight conditions in Chapter 7. Afterwards, Chapter 8 describes a proof-of-concept for an online adaptation method for the evolved population of controllers. Furthermore, Chapter 9 summarises the verification and validation process.

The report concludes with the fourth part, by reflecting on the research questions, their answers and contributions in Chapter 10. Last but not least, Chapter 11 highlights the proposals made for future steps in this line of research.

# Part I

## Scientific Article

# Evolutionary Reinforcement Learning: A Hybrid Approach for Safety-informed Intelligent Fault-tolerant Flight Control

V. Gavra* and E. van Kampen†
*Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands*

**Recent research in bio-inspired artificial intelligence potentially provides solutions to the challenging problem of designing fault-tolerant and robust flight control systems. The current work proposes SERL, a novel Safety-informed Evolutionary Reinforcement Learning algorithm, which combines Deep Reinforcement Learning (DRL) and neuro-evolutionary mechanisms. This hybrid method optimises a diverse population of non-linear control policies through both evolutionary mechanisms and gradient-based updates. We apply it to solve the attitude tracking task on a high-fidelity non-linear fixed-wing aircraft model. Compared to a state-of-the-art DRL solution, SERL achieves better tracking performance in nine out of ten cases, remaining robust against faults, changes in initial conditions and external disturbances. Furthermore, the work shows how evolutionary mechanisms can balance performance with the smoothness of control actions, a feature relevant for bridging the gap between simulation and deployment on real flight hardware.**

***Keywords:*** **Fault-tolerant Flight Control, Deep Reinforcement Learning, Evolutionary Algorithms, Intelligent Control, Control Policy Noise.**

## I. Introduction

Aviation has experienced an excellent safety record in recent decades, its ever-lowering rate of crashes making it the safest means of long-distance transportation in terms of the number of casualties. Despite this, the number of accidents involving 'loss of control in flight' is not decreasing at a comparable rate [1]. Improving the autonomy capabilities of aerospace systems in general and of aircraft controllers, in particular, could further cut the number of air crashes. Arguably, autonomy shall strive to become a reliable safety net acting at multiple levels of the aerospace industry.

Breakthroughs in intelligent control systems allow for increased autonomy and better human-machine teaming. There is hope that modern techniques incorporating bio-inspired Artificial Intelligence (AI) could enhance the safety, durability, autonomy, and performance of aerospace vehicles. Ultimately, a milestone in achieving these goals concerns improving the way simulation-based flight control systems are able to remain robust to real-life conditions such as faults, unmodeled dynamics and unforeseen disturbances [2–4].

Reinforcement Learning (RL) is a framework that uses repeated rewarded interactions between an intelligent agent and its environment to optimise its control policy [5]. Its extension, Deep Reinforcement Learning (DRL) inherits the advances from the field of deep learning (DL), which harvests the potential of deep neural networks (NNs) as data-driven models [6]. It provides a relatively sample-efficient way to train highly non-linear functions to approximate and control complex dynamics [5, 6]. These algorithms belong to the class gradient-based optimisation as the neural networks are trained via gradient descent [6].

Training on advanced computational systems, DRL has shown above-human performance in complex tasks such as strategy games [7, 8]. In flight control, it has obtained state-of-the-art performance in tasks considered difficult for more traditional control systems without any prior knowledge of the environment or solution. These include reference tracking in navigation using Deep Deterministic Policy Gradient (DDPG) [9] or Soft Actor-Critic (SAC) [10] and low-level attitude control in the presence of faults [11] (using SAC) or disturbances using a Proximal Policy Optimisation (PPO) agent [12]. The offline model-free DRL methods prioritise performance, often learning policies that command actions with high levels of noise. By doing so, they aggravate the gap between simulation and reality, increasing the challenge of deployment on flight hardware [13]. Regularisation methods have been developed to inhibit the action noise when controlling multi-rotor vehicles [13]. The root cause of the noise has yet to be extensively discussed.

*MSc. Student, Faculty of Aerospace Engineering, Control and Simulation Division, Delft University of Technology.
†Assistant Professor, Faculty of Aerospace Engineering, Control and Simulation Division, Delft University of Technology.

A rival of gradient-based policy optimisation, an Evolutionary Algorithm (EA) takes inspiration from nature's way of drafting complex and diverse organisms capable of sustaining life [14]. It iteratively optimises a set of control strategies, or population, by selecting the fittest (i.e., best performing) individuals in each generation and performing variations on them [15, 16]. As frameworks based on the evolutionary structure, recent advances using Genetic Algorithms (GAs) or Evolutionary Strategies (ES) made them attractive for continuous control tasks [14, 17]. Following this structure, neural evolution optimises control policies parameterised by neural networks, an alternative to the gradient-based DRL [18, 19].

Local optimisation by EAs results in a repertoire of behaviours, instead of a single policy. When evolving randomly initialised non-linear policies, the repertoire can benefit from intrinsic diversity as each individual learns to solve the task in a way novel from the other members of the population [14]. When deployed, the agent can then select the best-performing control policy either by estimating the performance using a model of the environment dynamics [20] or by learning online a locally-accurate approximation of the world dynamics [21, 22]. Such methods have been applied to adaptive legged robots [14], validating the transfer of behaviours emerging from offline population-based optimisation to real-life conditions.

Despite their evident achievements, both the aforementioned bio-inspired frameworks have their drawbacks. Gradient-based DRL algorithms have brittle convergence properties, which are highly sensitive to the chosen hyperparameters [23, 24]. They can also exhibit inefficient learning when faced with sparse gradients or deceptive optima (i.e., local minima) in the loss landscape [25]. Whereas GAs and ES can overcome these problems, they are less sample-efficient because they rely on generational updates of the entire population [26, 27].

Hybrid algorithms, incorporating evolutionary loops and reinforcement learning updates, show more stable learning and increased performance in complex learning environments [28–30]. The historically first Evolutionary Reinforcement Learning (ERL) algorithm, proposed by [24], trains a population of control policies in an evolutionary loop, periodically infusing it with DRL information obtained from gradient updates. Whereas the algorithm aims at the best of both worlds, the average performance of the entire population remains unstable due to spontaneous catastrophic forgetting and it is sensitive to the hyperparameters [28, 30]. Multiple frameworks aim to correct this behaviour by using more sample-efficient ES algorithms [31], incorporating safety-aware mutations [28, 32] or developing collaborative mechanisms within the population [29].

Proximal Distilled Evolutionary Reinforcement Learning (PDERL) [28] benefits from a modular structure, which combines a DDPG agent with a GA. It reduces the chance of catastrophic forgetting via proximal mutation and distillation crossover, thus matching the state-of-the-art performance in terms of maximum return [28, 30].

The current paper has a threefold contribution. First, we develop and apply a novel hybrid and safety-informed intelligent controller to the task of attitude tracking. Second, the paper shows that optimising a population of controllers via both DRL and GAs provides significantly better fault tolerance and robustness to unseen flight conditions compared to a state-of-the-art DRL-only approach. Lastly, we demonstrate that control policy noise can be alleviated by leveraging the evolutionary mechanisms of the hybrid method, shining a light on the root causes of the noise phenomena.

## II. Background

This section introduces the mathematical formalism of ERL by summarising the actor-critic RL, population-based policy search concepts and the hybrid algorithm combining the two. Additionally, it presents the attitude control task converted to a learning problem suitable for the ERL framework.

### A. Deep Reinforcement Learning

Reinforcement Learning is a bio-inspired framework that uses repeated rewarded interactions between an intelligent agent and its environment to learn an optimal control policy. The sequential decisions made by the agent within the environment are modelled as a Markov Decision Process (MDP) formalised by $\mathcal{P}\left\{\mathbf{s}_{t+1}, \tilde{r}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t, \ldots, \mathbf{s}_0, \mathbf{a}_0\right\} = \mathcal{P}\left\{\mathbf{s}_{t+1}, \tilde{r}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right\}$, with state space $\mathcal{S} \subset \mathbb{R}^n$, action-space $\mathcal{A} \subset \mathbb{R}^m$, reward signal $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ and probability distribution of the stochastic state transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The MDP, therefore, assumes collapsed state-action history as the current state and action solely determine the next state and received reward [5, 33].

For deterministic policies, at each time step, the agent commands action $\mathbf{a}_t \in \mathcal{A}$ according to a deterministic function of its state $\mu(\mathbf{s}_t)$ and observes the transition tuple $\mathcal{T}_t = \langle \mathbf{s}_t, \mathbf{a}_t, \tilde{r}_t, \mathbf{s}_{t+1} \rangle$. The sequence of transitions continues until the agent reaches a terminal state, marking the end of an episode.

The *return*, defined as $R_t = \lim_{n\to\infty} \sum_{k=0}^{n} \gamma^n \tilde{r}_{t+k}$, represents the total accumulated reward expected from time $t$

and discounted by $\gamma$, bounded by $(0, 1]$ for the summation to converge. Thus, the goal of the agent translates into maximising $R$. The action-value function $Q(\mathbf{s}, \mathbf{a})$ predicts the value of the return from the given state $\mathbf{s}$ when the agent selects action $\mathbf{a}$ thereafter following policy $\mu$ [5].

Policy gradient methods frame the goal of maximising return as the minimisation of a loss function $\mathcal{L}(\theta)$, updating a parametric approximation $\mu(\mathbf{s})_\theta$ (with $\theta \in \mathcal{D}_\theta$) of the optimal policy function $\mu^*(\mathbf{s})$. In contrast, value-based frameworks approximate the state-action value function as $Q(\mathbf{s}, \mathbf{a}) \approx Q_\phi(\mathbf{s}, \mathbf{a})$ with $\phi \in \mathcal{D}_\phi$, iteratively updating it to approximate the return value. Actor-critic methods combine policy and value function learning [5]. A state-of-the-art method, Deep Deterministic Policy Gradient employs neural networks as function approximators for both the critic and actor due to their intrinsic ability to learn complex non-linear dynamics from data [6]. DDPG, as a model-free RL algorithm, can solve continuous control tasks in complex flight control [9].

Despite this, a common failure mode for DDPG stems from the *overestimation bias* of the Q-function [34] which accumulates due to the recursive temporal difference updates. Over multiple updates, the error causes arbitrary sub-optimal states to be associated with a high Q-value, resulting in inconsistent policy updates and divergent behaviour [35]. The Twin-Delayed DDPG (TD3), developed by Fujimoto et al. [35], directly addresses the issue of Q-value overestimation.

Sustained exploration within the state-action space is a necessary condition for learning the optimal policy. In TD3, the behavioural policy explores during training by altering the actions selected by the actor, such that $\mathbf{a}_t = \mu(\mathbf{s}_t) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma I)$ is a sample of a zero-mean Gaussian distribution with deviation $\sigma$, the magnitude of the exploratory noise. In practice, the resulting action is clipped within the bounds admissible by the environment [35, 36]. After an action is taken in the environment, the transition tuple $\mathcal{T}_t$ is saved into a *memory buffer* $\mathcal{D}$.

During training, the networks are updated by randomly sampling batches $B$ from the buffer $\mathcal{D}$. The critic part trains using recursive temporal difference updates, minimising the Mean Squared Bellman error (MSBE) from Eq. (1). TD3 learns two Q-functions (hence its name) and it takes the minimum of the two Q-values as targets in the MSBE loss - the *duelling critics* trick. By involving the less optimistic value function as an optimisation target, the algorithm is less prone to overestimation bias which has been shown to improve learning stability [35].

$$\mathcal{L}_Q(\phi, B) := \frac{1}{|B|} \sum_{\mathcal{T}_t \in B} \left[ \left( Q_\phi(\mathbf{s}_t, \mathbf{a}_t) - \underbrace{\left( \tilde{r}_t + \gamma \min_{i=1,2} Q_{\phi_{i,\text{targ}}} \left( \mathbf{s}_t, \mathbf{a}_{\text{targ}_t}(\mathbf{s}_t) \right) \right)}_{y_{\text{targ}}(\tilde{r}, \mathbf{s}_{t+1})} \right)^2 \right] \qquad \phi \in \mathcal{D}_\phi, \theta \in \mathcal{D}_\theta \tag{1}$$

Concurrently, the actor learning translates into the minimisation of policy loss, defined by Eq. (2) as the negative of the value estimate over one batch of states.

$$\mathcal{L}_\mu(\theta, B) := -\frac{1}{|B|} \sum_{\mathbf{s}_t \in B} \min_{i=1,2} Q_{\phi,i} \left( \mathbf{s}_t, \mu_\theta(\mathbf{s}_t) \right) \tag{2}$$

The target networks are separate copies of the actor and critic networks, parametrised within the same $\mathcal{D}_\phi, \mathcal{D}_\theta$ spaces. To ensure stability during learning, they are synchronised using a Polyak average of the parameters of the current networks [36, 37].

TD3 also employs *clipped target actions* to train the target critic. Eq. (3) shows how the target-policy selected action is corrupted by noise sampled from a zero-mean Gaussian distribution (the same used for off-policy exploration) but clipped within $\left[c_{\text{low}} = -0.5, c_{\text{high}} = 0.5\right]$ as a form of regularisation. The resulting action is clipped again to lie within the interval admissible by the environment.

$$\mathbf{a}_{\text{target}} = \text{clip}\left( \mu_{\theta_{\text{targ}}}(\mathbf{s}) + \text{clip}(\boldsymbol{\epsilon}, c_{\text{low}}, c_{\text{high}}), \mathbf{a}_{\text{low}}, \mathbf{a}_{\text{high}} \right), \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma I) \tag{3}$$

Lastly, the target policy network updates less frequently than the target critic network. Adding to the smoothening effect of the Polyak update, delaying the policy updates further reduces the risk of divergence.

In DRL frameworks, the networks are updated by stochastic gradient descent (SGD), which requires the backpropagation of the loss functions' gradients with respect to the networks' parameters [6]. By doing so, the batch SGD updates drive sample-efficient experience-based training. Despite this, local optima and saddle points in the policy loss landscape alongside noisy or sparse gradient estimates impede long-term exploration and hinder learning [27].

## B. Evolutionary Algorithms

EAs are another bio-inspired alternative which, as a black-box local-search technique, iteratively improve a *population* of control policies or individuals targeting a higher *fitness* value [38].

EAs are characterised by a standard underlying architecture: evaluate the episodic performance of each individual, select the best-behaving individuals (fittest policies) and apply variational operators. A sub-class, Genetic Algorithms (GAs) are derivative-free optimisation algorithms that aim to deliver a sufficiently accurate solution within a relatively low computational time [39]. Having a modular structure, traditional GAs explore the solution space via two separate means: *crossover* and *mutation*. Crossover combines a fraction of elites and mutation alters the non-elites, both generating new offsprings to replace the least fit individuals.

In contrast to a gradient-based method, they do not have a tendency to find a local optimum but iteratively perform local searches for the global optimal fitness [39]. Moreover, according to [40], GAs are suited for both non-Markov and Markov decision processes, performing well in tasks that suffer from a lack of observability.

The selection operation is probabilistic according to the tournament selection method [29]. Solutions with higher fitness values have a higher probability of being selected. One assumes higher fitness values to represent better solution quality. With each generation, the average quality of the population improves, as well as the reward of the best actor, called the population *champion*.

## C. Hybrid Frameworks

Learning emerges from the trade-off between exploitation or quality-seeking and exploration, or novelty-seeking [14, 41]. ERL, introduced by [24], is a class of hybrid methods that implements this trade-off as a combination of off-policy DRL and GA. It evolves a population of $N$c control policies, applying selective pressure based on their episode fitness. After each episode lasting $T$ time-steps, the individual's fitness $F_i$ is defined via Eq. (4)

First, the GA part generates novel individuals, or *offsprings* through $K$-point crossover and Gaussian mutation. They both act on the *genome*, the NN weights tensor flattened to a one-dimensional sequence. $K$-point crossover combines genetic material from two policies by dividing their genomes into $K$ randomly sampled sections and simply swapping them. The standard Gaussian mutation, showed by Eq. (5) samples the offspring policy weights $\theta_o$ from a Gaussian distribution centred at the parent weights vector $\theta_p$.

$$F_i := \sum_{t=0}^{T} \tilde{r}_t, \quad i = \overline{1, N} \qquad (4) \qquad\qquad \theta_o \sim \mathcal{N}\left(\theta_p, \sigma_{mut} I\right) \qquad (5)$$

Second, parallel to the genetic population, an actor-critic agent based on DDPG learns via gradient updates by randomly sampling batches from the common memory buffer. Training is performed off-policy, as the actor and critic are updated with information gathered by all the other policies. At a given number of generations, the RL actor is injected into the population by cloning its weights in place of the least-fit genetic actor. Thus, ERL establishes a bidirectional transfer of information [24, 28]. Whereas the shared memory buffer enhances exploration of the state space off-policy, the actor synchronisation aims to decrease the sample-complexity of the neuro-evolution as the newly injected NN has been updated more often via gradient-based optimisation [24, 30].

The hybrid agent sequentially evaluates all actors and stores the experienced transition tuples in a *shared memory buffer* $\mathcal{D}^{(N)} := \left\{\mathcal{T}_t^{(i)} \mid i = \overline{1, N}\right\}$. The superscript (N) refers to the size of the genetic population and will be later dropped for brevity. Thus, the ERL agent's shared buffer contains unordered exploratory traces of both GA and DRL loops.

Whereas the DDPG actor uses zero-mean additive Gaussian noise to explore in the $\mathcal{S} \times \mathcal{A}$ space, ERL combines this with indirect time-independent exploration in the policy parameter space $\mathcal{D}_\theta$ [27, 42]. Crucially, since the distribution of the weight-magnitude is not uniform across the network, the output of each layer is normalised to more evenly scale the effect of the applied perturbations [42].

Directly extending the ERL method from [24], Proximal Distilled Evolutionary Reinforcement Learning (PDERL) developed by Bodnar et. al. uses the same structure as ERL and a DDPG agent, but it proposes two novel genetic operators, distillation crossover and proximal mutation which sample experiences from individual buffers of the genetic actors [28]. They aim to counteract catastrophic forgetting caused by Gaussian mutation and $K$-point crossover applied to a direct encoding of the network, a problem of the off-policy hybrid methods [28, 30, 32].

### *1. Genetic memory buffer*

To integrate both operators, PDERL uses a set of memory buffers with each actor possessing *genetic memory* $\mathcal{D}^G$. The individual buffers store the $\kappa$ most recent experiences of each actor. Moreover, each personal buffer can store experiences that span multiple generations which will be spread through the population by the variational operators. The common buffer $\mathcal{D}$ shares experience with the genetic memories but does not necessarily equal the union of them.

### *2. Q-filtered distillation crossover*

The operator selectively merges the behaviours of two elite policies based on their estimated Q-values, acting based on the behavioural space learnt by the critic [28]. Essentially, offsprings inherit optimised behaviours from their parents and not just the weights of the networks.

Consider two distinct parent policies $\mu_x$ and $\mu_y$. An offspring policy $\mu_o$ is created and its buffer $\mathcal{D}_o$ is filled with $\kappa/2$ experiences from both parents. Its weights $\theta_o$ are randomly initialised with the weights of one of the parents. Then, the offspring actor is optimised to selectively imitate its parents' actions for the sampled states. In general, this form of *imitation learning* [43], 'distils' the behaviour of the parents into the offspring policy. The footprint (i.e., the number of learnable parameters) of the resulting network is equal to the parents'. Thus, the footprint of each actor is kept constant.

Crossover combines two networks instead of one which introduces the problematic possibility of offspring's behaviour diverging from both parents. To account for it, PDERL introduces a cloning loss $\mathcal{L}^{(C)}$ which trains the offspring $\mu_o$ to minimise the deviation from the parents' behaviours over a batch $B$:

$$\begin{aligned}\mathcal{L}_{\mu}^{(C)}(\theta_o, B) := &\sum_{i}^{|B|} \|\mu_o(\mathbf{s}_i) - \mu_x(\mathbf{s}_i)\|^2\, \mathbb{I}_{Q(\mathbf{s}_i,\mu_x(\mathbf{s}_i))>Q(\mathbf{s}_i,\mu_y(\mathbf{s}_i))} \\ &+ \sum_{j}^{|B|} \left\|\mu_o(s_j) - \mu_y(\mathbf{s}_j)\right\|^2 \mathbb{I}_{Q(\mathbf{s}_j,\mu_y(\mathbf{s}_j))>Q(\mathbf{s}_j,\mu_x(\mathbf{s}_j))} \\ &+ \frac{1}{|B|}\sum_{k}^{B_C} \|\mu_o(\mathbf{s}_k)\|^2 \quad \text{with} \quad B \sim \mathcal{D}^o\end{aligned} \tag{6}$$

The operator $\mathbb{I}$ selects the policy with a higher Q-value, biasing the offspring towards the estimated best-acting parent. The last term of Eq. (6) performs $L_2$-norm regularisation with the intention to optimise for less aggressive behaviours (i.e., actions changing rapidly for marginally different states) [28].

Lastly, one still has to decide the elites whose weights are going to be crossed. The pairing metric $d : \mathcal{S} \rightarrow \mathbb{R}$ from Eq. (7) is the Euclidean distance between the average actions taken by the agents [28]. It takes a randomly sampled batch of states from the parent's genetic memory with $\mathcal{P}_{x,y}$ denoting the generic state-visitation probability distribution.

$$d\left(\mu_x, \mu_y\right) := \mathbb{E}_{\mathbf{s}\sim\mathcal{P}_x}\left[\left\|\mu_x(\mathbf{s}) - \mu_y(\mathbf{s})\right\|^2\right] + \mathbb{E}_{\mathbf{s}\sim\mathcal{P}_y}\left[\left\|\mu_x(\mathbf{s}) - \mu_y(\mathbf{s})\right\|^2\right] \tag{7}$$

The probability of selecting a pair increases with the distance $d\left(\mu_x, \mu_y\right)$ between them. This distance-based selection strategy incentives behavioural *novelty*, developing a diverse population. Given the distance metric definition, the achieved novelty is acts across the space mapped by the actuator deflections.

### *3. Proximal mutation*

Let one consider the parent to be mutated $\mu_p$, parametrised by $\theta_p$ with genetic memory $\mathcal{D}_p$. The mutation operator samples a batch of transitions $B_M$ and sums the policy gradient with respect to its parameters over the states within $B_M$. Eq. (8) computes the policy sensitivity $s_{\mu_p}$ as the root of the squared gradient summed over the action space:

$$s_{\mu_p} := \sqrt{\sum_{k}^{|\mathcal{A}|}\left(\sum_{i}^{|B_M|} \nabla_\theta \mu_\theta(\mathbf{s}_i)\right)_k^2}, \quad B_M \sim \mathcal{D}^p \tag{8}$$

Then, the sensitivity is used to scale the zero-mean Gaussian perturbation, the policy parameters being updated according to $\theta_o \sim \mathcal{N}\left(\theta_p, \frac{\sigma_{mut}}{s} I\right)$. The higher the magnitude of the policy gradient with respect to a parameter, the smaller the mutation step becomes. Essentially, sensitive NN parameters are perturbed less. After the scaled update, the resulting offspring policy $\mu_o$ has its parameters $\theta_o$ within the proximity of its predecessor. Thus, the offspring's behaviour is expected to not diverge too much either.

## III. Methodology

This section introduces the attitude tracking task and then presents the framework of Safety-informed Evolutionary Reinforcement Learning (SERL), an extension obtained by including the TD3 algorithm inside PDERL and accounting for safety-related information in the mutation operator. Lastly, the training and evaluation scenarios are described.

### A. Attitude Tracking

The simulation environment for learning and evaluation considers continuous attitude control of a fixed-wing aircraft model. The model with high-fidelity non-linear plant dynamics is compiled as a standard Flight CAD package referred to as DASMAT [44]. It was designed by [44] within the Control & Simulation department of the TU Delft Faculty of Aerospace Engineering to model the Cessna 500 aircraft. Later, it has been improved by [45] through system identification on flight test data recorded on-board the PH-LAB, the research aircraft jointly owned between the TU Delft Faculty of Aerospace Engineering and Netherlands Aerospace Centre (NLR) [*].

The model incorporates fixed-wing aerodynamics, propulsion and engine, resulting in a 6-DOF combined translational and rotational nonlinear equations of motion for the rigid body aircraft, trimmed for the specific flight conditions. The complete aircraft state $\mathbf{x} \in \mathbb{R}^{12}$ is given by Eq. (9), where the longitudinal and lateral displacements $X_e$ and $Y_e$ are measured with respect to the trim point.

$$\mathbf{x} = [p, q, r, V_{tas}, \alpha, \beta, \theta, \phi, \psi, H, X_e, Y_e]^\top \tag{9}$$

The agent state vector $\mathbf{s} \in \mathbb{R}^7$, defined by Eq. 11 retrieves observed information from the aircraft state at a sampling rate $f_s = 100\,\text{Hz}$, augmenting it with the current tracking error. The actions $\mathbf{a}_t = \mu(\mathbf{s}_t) \in \mathbb{R}^3$ outputted by the behavioural policy are fed as the input vector at the same frequency $f_s$. Defined by Eq. 10, the actor's action corresponds to the deflection of the elevator, aileron and rudder. The thrust control is delegated to an inner auto-throttle [11, 44] and the trim tab and flap deflections are kept at zero. The control diagram from Fig. 1 shows the feedback loop between the non-linear intelligent agent and the controlled plant.

$$\mathbf{a} := [\delta_e, \delta_a, \delta_r]^\top \tag{10}$$

$$\mathbf{s} := [\theta_e, \phi_e, \beta_e, p, q, r, \alpha]^\top \tag{11}$$

To account for realistic physical limits of the actuators, the space $\mathcal{A}$ is restricted for each channel bounded to $[-10°, 10°]$. The ranges are within the admissible deflections of the modelled control surfaces [46].

The reference signals correspond to a random sequence of coordinated pitch-roll manoeuvres. Following from [11] and [47], the $\theta_r$ and $\phi_r$ references are sequences of cosine-smoothed step signals with amplitudes uniformly sampled from the ranges: $[-25°, 25°]$ for pitch and $[-45°, 45°]$ for roll. The latter is motivated by the values specified in the CS-25 [48]. The range for pitch is based on the previous work of [11, 47, 49]. The sideslip reference remains at zero.

For learning, the control task is translated into an optimal control formulation with a continuous reward signal from Eq. (12), taken from [11]. The reward function penalises the $L_1$ norm of the clipped and scaled attitude tracking error. The scaling factor $\mathbf{c}_r = \frac{6}{\pi}[1, 1, 4]$ accounts for the smaller magnitude of sideslip error.

$$\tilde{r}_t = -\frac{1}{3} \left\| \text{clip}\left[\mathbf{c}_r \odot \mathbf{e}_t, -1, 0\right] \right\|_1 \quad \text{with} \quad \mathbf{e}_t := \left[\theta_{r_t} - \theta_t, \phi_{r_t} - \phi_t, \beta_{r_t} - \beta_t\right]^\top \tag{12}$$

At the end of each episode, the agent receives an additional sparse negative reward $\tilde{r}_T$ which penalises it proportionally to the time frames left before the prescribed episode should have ended. In Eq. (13), the positive constant $C_P$ scales the negative consequences of crashing early. Empirically, a value of $C_P = 20$ is deemed appropriate.

$$\tilde{r}_T = -\frac{C_P}{\Delta t}\left(T_{max} - T\right) \tag{13}$$

### B. Learning Framework

The Safety-informed Evolutionary Reinforcement Learning referred to as SERL, extends PDERL by replacing the DDPG actor-critic with Twin Delayed Deep Deterministic Gradient (TD3) and the proximal mutation operator with the safety-informed mutation operator.

[*]https://cs.lr.tudelft.nl/citation/

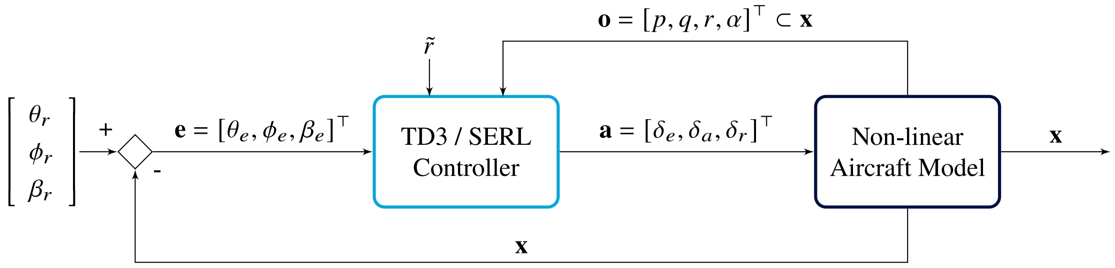

$\mathbf{o} = [p, q, r, \alpha]^\top \subset \mathbf{x}$
$\tilde{r}$
$\theta_r$ $\phi_r$ $\beta_r$
+
-
$\mathbf{e} = [\theta_e, \phi_e, \beta_e]^\top$
TD3 / SERL Controller
$\mathbf{a} = [\delta_e, \delta_a, \delta_r]^\top$
Non-linear Aircraft Model
$\mathbf{x}$
$\mathbf{x}$

**Fig. 1 TD3 / SERL attitude control diagram.**

*1. Framework overview*

SERL has a modular design, where each component can be developed separately. The additional argument ($N$) denotes the number of actors of the genetic population, with SERL($N$) having a total policy count of $N + 1$ due to the extra TD3 actor. Fig. 2 depicts the high-level overview of its components and the interaction between the RL agent (blue), the GA loop (green) and the simulation environment (grey).

Directly improving on DDPG, TD3, a deterministic, offline and off-policy method, improves the sample complexity and learning stability in continuous control [35, 50]. Its achievements and relatively uncomplicated structure made TD3 the leading candidate for the RL loop of the current SERL framework.

SERL uses two types of replay buffers. The shared buffer trains the RL agent in an off-policy manner but each actor within the population has its own genetic buffer. The critics are feedforward neural networks with different shapes and sizes than the genetic actors and TD3 policy, which have the same footprint.

Empirically, it has been observed that $N$ is the hyperparameter with the highest impact on training, directly altering the balances between the GA and TD3 learning. A larger population increases the number of frames experienced between RL updates and it reduces the number of transitions corrupted with exploratory noise. Thus, increasing the genetic population limits the RL effort. At the same time, both the crossover and mutation operators become more effective at developing novel behaviours once the genetic pool increases.

*2. Safety-informed mutation*

The hybrid algorithm structure offers the opportunity to develop either of the optimisation loops with overall potential gains. However, to take into account existing safety-related domain knowledge, a safety-informed mutation operator [32] is adopted.

Similarly to the proximal mutation, this safety-informed operator applies noise scaling proportionally to the inverse of the policy gradient norm as previously shown in Eq. (8). The policy gradients are estimated on batches $B_C$ sampled from the parent's *critical buffer* $\mathcal{D}_C \subset \mathcal{D}$. It contains the transition tuples associated with a high value of a cost function, directly related to safety knowledge on the manoeuvre.

Eq. (14) defines the SERL critical buffer as a sub-set of transitions that cause the next-state angle of $\alpha$ and $\phi$ approach a set of simplified flight envelope bounds. These are marked by entering pre-stall dynamics at an angle of attack higher than 11° or by having a roll angle larger than 60°, empirically observed to degenerate in an unstable spiral motion.

$$\mathcal{D}_C := \{\langle \mathbf{s}_t, \mathbf{a}_t, \tilde{r}_t, \mathbf{s}_{t+1} \rangle \in \mathcal{D} | \quad \alpha_{t+1} \geq 11^\circ \quad || \quad \phi_{t+1} \geq 60^\circ\} \tag{14}$$

The safety-informed mutation overrides the effect of the proximal operator and directs exploration towards the less-sensitive regions of the parameters space $\mathcal{D}_\theta$. Essentially, the offspring's genome will evolve less in the directions that could result in unsafe behaviours. Nevertheless, since the operator scales the additive parameter noise, unsafe exploration is only discouraged and not stopped so the resulting policies do not benefit from safety guarantees.

## C. Optimisation for Policy Smoothness

To combat policy noise, the Conditioning for Action Policy Smoothness (CAPS) regularises the policy loss function according to Eq. (15). Whereas the term $\mathcal{L}_T$ aims for Lipschitz temporal continuity by minimising the $L_2$ norm between the current action and the next action, the spatial term $\mathcal{L}_S$ penalises noise in system dynamics by ensuring a locally consistent policy [13]. The regularisation weights, $\lambda_T$ and $\lambda_S$, control the strength of the corresponding terms and

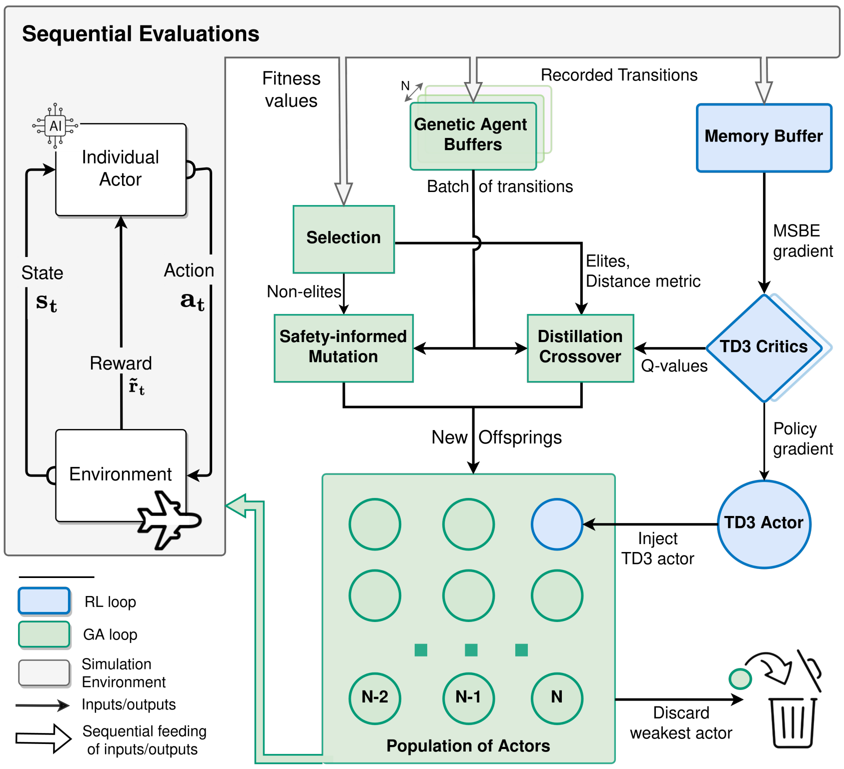


**Fig. 2 High-level overview of the SERL($N$) framework. Safety-informed mutation only considers the critical transitions from the genetic memory. Solid single arrows show data flow and double arrows depict the sequential feeding of information, as each actor within the population is evaluated separately inside the environment. Inspired from [24, 28, 32].**

can be tuned. During training, CAPS adds their weighted sum to the loss function from Eq. (2) and the SGD updates propagate the effect of $\mathcal{L}$ terms. By doing so, the RL agent can learn a smoother control policy.

$$\begin{aligned}\mathcal{L}_S &= \|\mu_\theta(\mathbf{s}) - \mu_\theta(\tilde{\mathbf{s}})\|_2 \quad \text{with} \quad \tilde{\mathbf{s}} \sim \mathcal{N}(\mathbf{s}, \sigma I)\\ \mathcal{L}_T &= \|\mu_\theta(\mathbf{s}) - \mu_\theta(\mathbf{s}_{t+1})\|_2 \\ \mathcal{L}_\mu^{(\text{CAPS})} &= \lambda_S \mathcal{L}_S + \lambda_T \mathcal{L}_T\end{aligned} \tag{15}$$

Due to their inherent non-linearity, neural networks policies output action signals with spectral components in which the input information is not readily visible. To objectively measure and compare the smoothness of different policies following the same trajectories, the $Sm$ metric from Eq. (16) takes the frequency-weighted sum of the spectral amplitudes of all actuating signals. The metric is inspired by the smoothness defined in [13] but with two key differences. Dividing by the episode duration $T$ penalises shorter trails, hence $Sm$ reflects an average smoothness. The square root scales down the relative smoothness, thus the unit of the $Sm$ metric is rad · Hz. Lastly, the negative sign ensures that higher $Sm$ corresponds to smoother actuating signals and the positive scaling constant $C_{S_m}$ (set to 10) makes the metric arbitrarily comparable to the reward-based fitness term already defined in Section II.

$$Sm := -\frac{C_{S_m}}{T} \cdot \sqrt{\frac{2}{n} \sum_{a=1}^{|\mathcal{A}|} \sum_{k=1}^{n/2} \left(S_{aa}[k] \cdot f_k\right)_a} \tag{16}$$

The spectral amplitude $S_{aa}[k] = \frac{1}{n}|A[k]|^2$ is estimated with the discrete Fourier Transform $A[k]$ of the actuating signal $a \in \mathcal{A}$ calculated via the Fast Fourier Transform (FFT). The spectral components are summed up to the discrete equivalent of $f_s/2$, the Nyquist frequency of the actuator signal sampled at $f_s$.

While CAPS indirectly optimises spatial and temporal continuity, it is more difficult to objectively quantify action smoothness in multi-dimensional spaces. Moreover, whereas RL learns from frames, SERL performs updates after episodes are completed. Therefore, $Sm$ is not only used to evaluate the trained actors of both TD3 and SERL but also as an additional fitness term for evolutionary selection.

For the training scenario described by Section III.D, the fitness is defined according to Eq. (4). Additionally, an ablation study investigates the effect of CAPS regularisation in comparison to $Sm$ optimisation. The latter is obtained by adding the $Sm$ term from Eq. (16) to the fitness function.

### D. Training Task

For the training environment, the aircraft model is trimmed for steady straight symmetric flight at $V_{tas} = 90\,\mathrm{m/s}$ and $H = 2,000$ m. The learning phase concerns repeated training epochs alternated with testing epochs at a user-specified frequency. During learning, the maximum duration of one training episode is $T_{max} = 20\,\mathrm{s}$, equivalent to $2,000$ agent-environment interactions. The training algorithm considers five agent-environment pairs, initialised with different seeds, and periodically evaluates each TD3 actor and champion policy to compute their episodic return statistics.

For SERL, choosing the episode length is a trade-off between actors receiving enough samples to explore and the training computational complexity. Longer episodes would increase the time required for the neuro-evolutionary loop. Training either TD3 or SERL(10) for $1 \times 10^6$ frames required approximately 200 min on 8 Intel Xeon E5-1620 3.50 GHz CPU cores.

Most of the hyperparameters configure the TD3 agent, apart from the last five parameters which trace back to PDERL settings. Among them, the mutation magnitude $\sigma_{mut}$ has the highest impact on exploration. The synchronisation rate controls the generational frequency of copying the TD3 policy weights. Also, the outcome of population evaluations dictates the champion so a large number of evaluations is more informative but also time-consuming. For SERL(50), using fewer evaluations helps reduce the training time while the number of training steps has been proportionally increased to account for the additional learning actors.

### E. Evaluation Tests

The policies optimised by each agent are tested on multiple scenarios, listed in Table 2. They are selected from the surveyed literature based on their relevance to flight systems, expected difficulty and possibility to be modelled within the DASMAT framework with the described attitude control architecture.

The cosine-smoothed step references follow similar coordinated pitch-roll manoeuvres but the evaluation episodes are longer, lasting $T_{max} = 80\,\mathrm{s}$. Each test case loops through the same list of pseudo-random references generated using three different seeds. Thus, each intelligent controller is evaluated on the same tasks to enable a fair comparison between their tracking performance and smoothness.

In Table 2, the nominal scenario considers the same flight conditions as the training task and it serves as the benchmark for the other cases to be compared against. The next six fault cases are based on the work of [11]. Simulating the agent's response with the controlled plant altered according to each of these cases investigates their fault-tolerant capabilities.

Cases R1 and R2 consider two flight conditions with different dynamic pressure by changing the airspeed and altitude from the trim setting used during training. Since the actuators and aircraft dynamics are affected by the dynamic pressure, these tests benchmark the agent's robustness to unseen flight regimes.

The last case tests the capacity to reject an external disturbance in the presence of realistic sensor models shown by Table 3, obtained from [51]). The simplified disturbance is a vertical gust of wind whose velocity is specified by the MIL-F-8785C specification [52].

The normalised Mean Absolute Error (nMAE) evaluates the tracking performance by averaging pitch, roll and sideslip. The normalisation interval is $[-25°, 25°]$ for the pitch and yaw channels whereas the sideslip error is normalised by $[-1°, 1°]$ as its response is expected to remain in the neighbourhood of zero.

**Table 1 Hyperparameters for the TD3, SERL(10) and SERL(50).**

| **Parameter Name** | **Symbol** | **TD3** | **SERL(10)** | **SERL (50)** |
|---|---|---|---|---|
| Population size | N | - | 10 | 50 |
| Learning rate (actor, critics) | $\alpha_{a,c}$ | $4.33 \times 10^{-4}$ | $4.82 \times 10^{-5}$ | $1.86 \times 10^{-5}$ |
| Discount factor | $\gamma$ | 0.99 | 0.99 | 0.99 |
| Batch size | $\vert B\vert$ | 64 | 86 | 256 |
| Buffer size | $\vert\mathcal{D}\vert$ | 100,000 | 800,000 | 2,000,000 |
| Actor hidden layers | $n_h^a$ | 3 | 3 | 3 |
| Actor hidden sizes | $h^a$ | (96, 96,96) | (72,72,72) | (72,72,72) |
| Critics hidden layers | $n_h^c$ | 2 | 2 | 2 |
| Critics hidden size | $h^c$ | (200,300) | (200,300) | (200,300) |
| Non-linear activation | - | ReLU | tanh | tanh |
| Exploratory noise magnitude | $\sigma$ | 0.33 | 0.29 | 0.23 |
| Mutation magnitude | $\sigma_{mut}$ | - | $2.47 \times 10^{-2}$ | $6.27 \times 10^{-2}$ |
| Elite Fraction | $e$ | - | 0.3 | 0.2 |
| Genetic memory size | $\kappa$ | - | 8,000 | 8,000 |
| Population evaluations | - | - | 5 | 3 |
| Synchronisation rate | - | - | 1 | 1 |

**Table 2 Evaluation cases for fault-tolerance and robustness.**

| **Identifier** | **Title** | **Description** |
|---|---|---|
| **-** | Nominal | Trim condition unchanged from training: $H = 2,000$ m, $V_{tas} = 90\,\mathrm{m/s}$ |
| **F1** | Iced Wings | Maximum angle of attack reduced by 30% and the drag coefficient increased with 0.06 |
| **F2** | Shifted Centre of Gravity | Aircraft centre of gravity shifted aft by 0.25 m |
| **F3** | Saturated Aileron | Aileron deflection clipped at ±1 deg |
| **F4** | Saturated Elevator | Elevator deflection clipped at ±2.5 deg |
| **F5** | Broken Elevator | Elevator effectiveness coefficient multiplied by 0.3 gain |
| **F6** | Jammed Actuator | Rudder stuck at a deflection of 15 deg |
| **R1** | High Dynamic Pressure | Trim condition at high dynamic pressure: $H = 2,000$ m, $V_{tas} = 150\,\mathrm{m/s}$ |
| **R2** | Low Dynamic Pressure | Trim condition at low dynamic pressure: $H = 10,000$ m, $V_{tas} = 90\,\mathrm{m/s}$ |
| **R3** | Disturbance & Biased Sensor-noise | Sensor models from Table 3, vertical wind gust of 15 ft/s acting for 3 s |

**Table 3 Gaussian noise sensor models identified from flight tests on the PH-LAB. The relevant observations have been isolated from [51].**

| Observed Signal | Unit | Noise Magnitude | Bias |
|---|---|---|---|
| $p, q, r$ | rad/s | $4.0 \times 10^{-7}$ | $3.0 \times 10^{-5}$ |
| $\theta, \phi$ | rad | $1.0 \times 10^{-9}$ | $4.0 \times 10^{-3}$ |
| $\alpha$ | rad | $4.0 \times 10^{-10}$ | - |

## IV. Results and Discussion

This section presents the results of learning, fault tolerance and initial condition robustness for both three distinct agents: SERL(10), SERL(50) and TD3. It also discusses the effects of involving direct *Sm* optimisation. If not specified otherwise, the sample average and standard deviation are computed over 30 random evaluations generated from three seeds. To verify whether the nMAE averages differed significantly, the *t*-test is used under the assumption of normally distributed nMAE samples.

### A. Learning Curves

The training curves from Fig. 3 depict the effect of policy learning by plotting the average episodic reward obtained by each policy alongside the total number of time steps the policies expired interacting with the environment. Ultimately, all agents converge towards returns, comparable to values reported by [11, 47, 49].

TD3 shows an initial advantage but suffers from unstable learning progress marked by spikes in standard deviation followed by sudden drops in the return, referred to as *catastrophic forgetting*. In contrast, both SERL agents are less sample efficient whereas their training progresses less erratically than TD3, and with a lower and monotonically decreasing deviation.

SERL(50) requires five times more samples to reach a similar return - a direct drawback of the poorer sample efficiency of genetic algorithms. SERL(10) learns with sample complexity comparable to the TD3 method as its gradient-updated actor is more often selected among the elites. Thus within SERL, TD3 learning drives early-stage learning, its elitism rate increasing up to 25% after $5 \times 10^5$ frames. Then, the rate of RL selection slowly decreases, marking the switch to genetic mechanisms which allow for long-term performance.

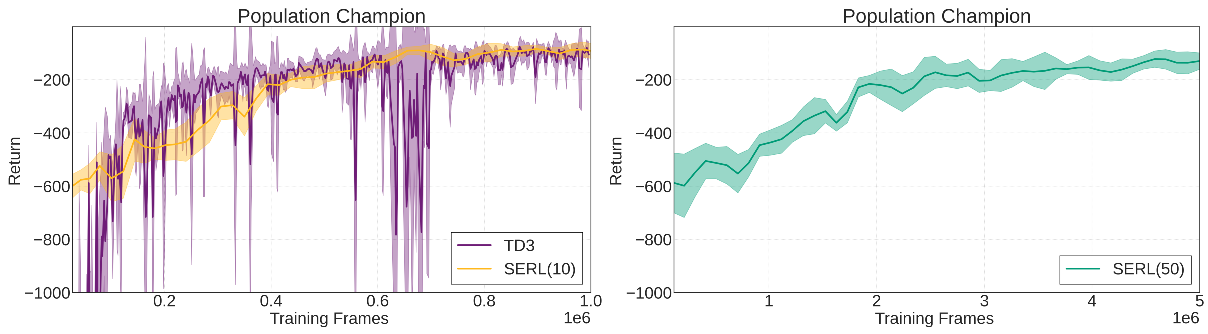


**Fig. 3 Learning curves of the TD3, SERL(10) and SERL(50) agents. The solid lines depict the average champion return and the shaded area denotes one standard deviation, computed over 5 mono-seed evaluations. For the TD3 agent, the champion is equivalent to the TD3 actor.**

### B. Fault-tolerancy

Despite validating the effect of training, learning curves do not paint a full picture of the control capabilities. Figure 4 shows the fault-case evaluation performance of the champion policy of SERL(10) and SERL(50) alongside the TD3-only agents trained in parallel to the population by the hybrid loop and the separately trained TD3. The best-behaving SERL(50) actor significantly outperforms ($p < 0.05$) both the SERL(10) champion and the TD3-only actor for all cases apart from the last two fault cases. Looking at F5 where SERL(50) obtains an nMAE score lower by 6.4% with $p = 0.1 > 0.05$ whereas for F6, the TD3-only actor tracks the references better than the best of SERL(10) and SERL(50) by 16.5% ($p = 2 \times 10^{-35} < 0.05$) and by 6.1% ($p = 8 \times 10^{-13} < 0.05$) respectively. In the last fault case, the nMAE scores are significantly higher than the rest due to the large and relatively constant sideslip bias (see Fig. 6).

For both hybrid methods, the identified champion for each case obtains an average tracking error at least as good as the TD3 actor trained alongside. In the case of a small population, the RL part drives the policy optimisation so the TD3 actor is more often selected within the elites. This happens for the nominal scenario and for the faults F3, F4 and F6 respectively. The bar chart shows that, in the context of the current control tasks, training multiple non-linear controllers benefit from the emerging diversity and can achieve higher tracking performance in almost all fault cases.

Next, F3 and F6 will be discussed, signifying the opposite ends on the relative performance spectrum.

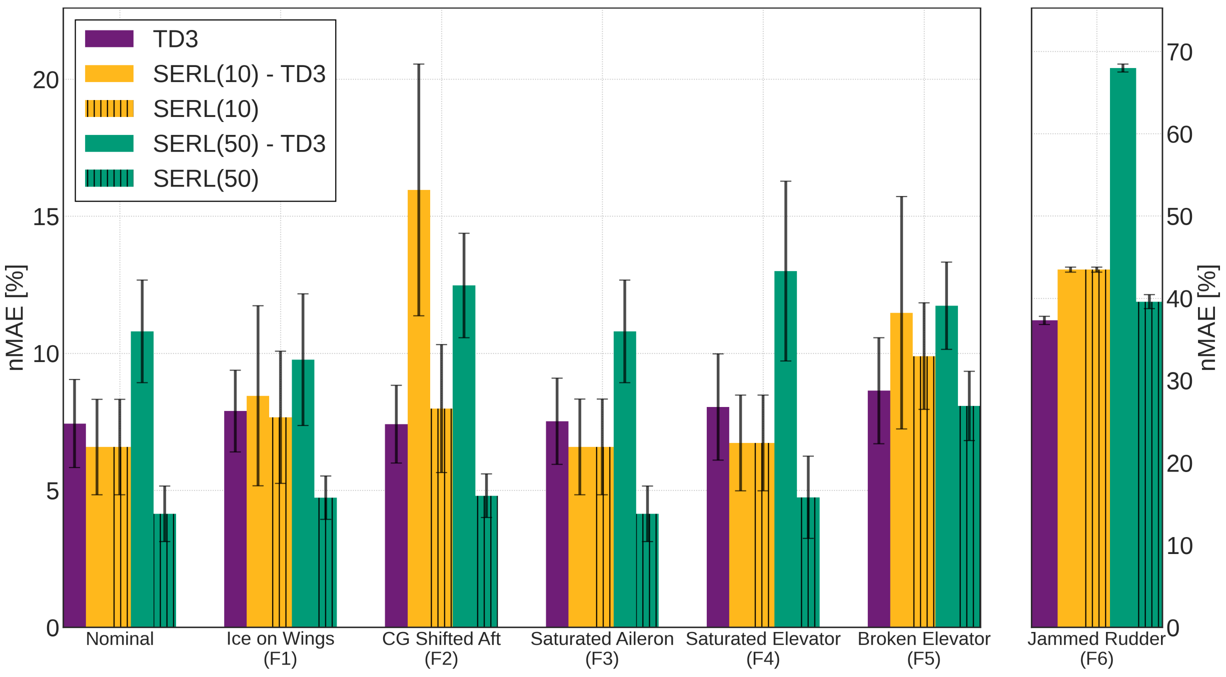


**Fig. 4 Tracking nMAE error and its deviation for each fault case. The hatched bars correspond to the population champion whereas the solid-coloured bars to the TD3 agent trained alone or within SERL.**

*1. Saturated Aileron (F3): downside of aggressive actions*

Figure 5 depicts the responses of the SERL(50) champion and the TD3 actor when tasked to control a plant with a clipped actuator, the third case from Table 2.

Large actuating bounds during training allow the policies to exploit them and control the system aggressively. In Fig. 5, comparing the responses computed by the two actors, the $\delta_a$ commanded by the SERL(50) champion is always within the saturation bounds. With a relatively calm response (i.e., low frequency and magnitude), the SERL(50) champion is able to follow the roll and sideslip references with the same accuracy as in the nominal case. It obtains an nMAE tracking error lower by 45% ($p = 4 \times 10^{-10} < 0.05$) than its RL-only counterpart.

The TD3 actor behaves more aggressively across all actuating channels, with high-frequency commands and large deflections. At $t = 20$ s, the roll angle reference shifts from 3° to −3°, prompting the TD3 actor to command a high aileron deflection. Since the simulated actuator is clipped, the TD3 actor pushes further, unaware of the fault. Thus, the commanded $\delta_a$ spikes, approaching the lower bound of $\delta_{a_{low}} = -10°$. Arguably, this unnecessary and sub-optimal behaviour stems from a preference for aggressive actions learnt by the TD3 agent.

*2. Jammed Rudder (F6): lost novelty*

When dealing with a rudder stuck at $\delta_r = 15°$, the genetic champions cannot match the error of the TD3-only agent. Figure 6 shows the responses of the aircraft when controlled by the SERL(50) champion (top plots) and the TD3 actor (bottom plots). One can argue that, when one channel is completely blocked, a significant amount of the behavioural diversity, encoded by the evolved population, had been lost. Thus, training multiple NNs is less advantageous than concentrating the frame-based learning updates on one single policy, as done by the TD3-only agent.

Also, while the SERL(50) champion tries to correct for the induced sideslip deviation, the TD3 actor does not. A potential explanation stems from the stronger coupling between the rudder and aileron channel of the TD3-only policy.

## C. Robustness

Previous works have shown robustness to the flight condition of a stochastic policy trained within a hierarchical SAC architecture and exposed to one trim condition [11, 49]. Despite this, their outer loop controller observed the altitude which offers partial information with respect to the dynamic pressure.

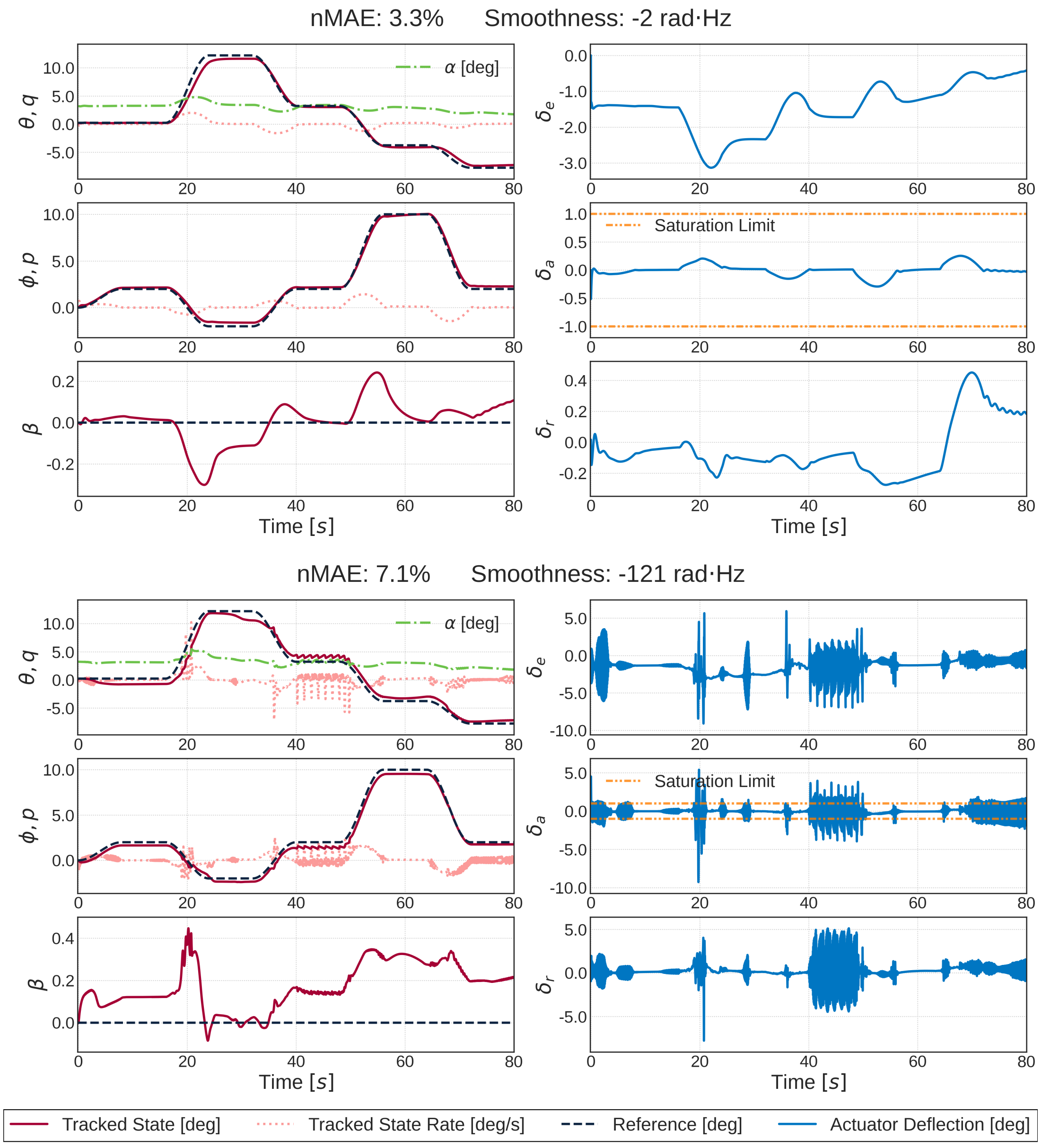


**Fig. 5 Evaluation time-traces of the SERL(50) champion (top) and TD3 (bottom) policy with their corresponding nMAE and $Sm$ metric for the damaged aileron (F3). The aileron saturation limit is shown as a separate interrupted line.**

We tested the new TD3 and SERL agent's robustness on two more extreme cases than the ones previously investigated. As attitude controllers observe the attitude reference, the corresponding error and the attitude rate, they have no direct knowledge of dynamic pressure. The bar chart from Fig. 7 shows the SERL(50) champion as the best-performing controller.

High dynamic pressures would increase the effectiveness of all three control surfaces considered. This benefits both of the SERL controllers, with the SERL(50) error deviation decreasing and the SERL(10) tracking the reference slightly better (14.2% lower error with $p = 0.06 > 0.05$). The TD3 actor experience the opposite, its R1 nMAE being higher by 35.4% ($p = 1 \times 10^{-7} < 0.05$) than the nominal case. Again, the TD3 policy is more aggressive, a sub-optimal trait when the controlled actuators become more effective.

Flying in low dynamic pressure makes the responses sluggish. Reacting to it proves problematic for all controllers

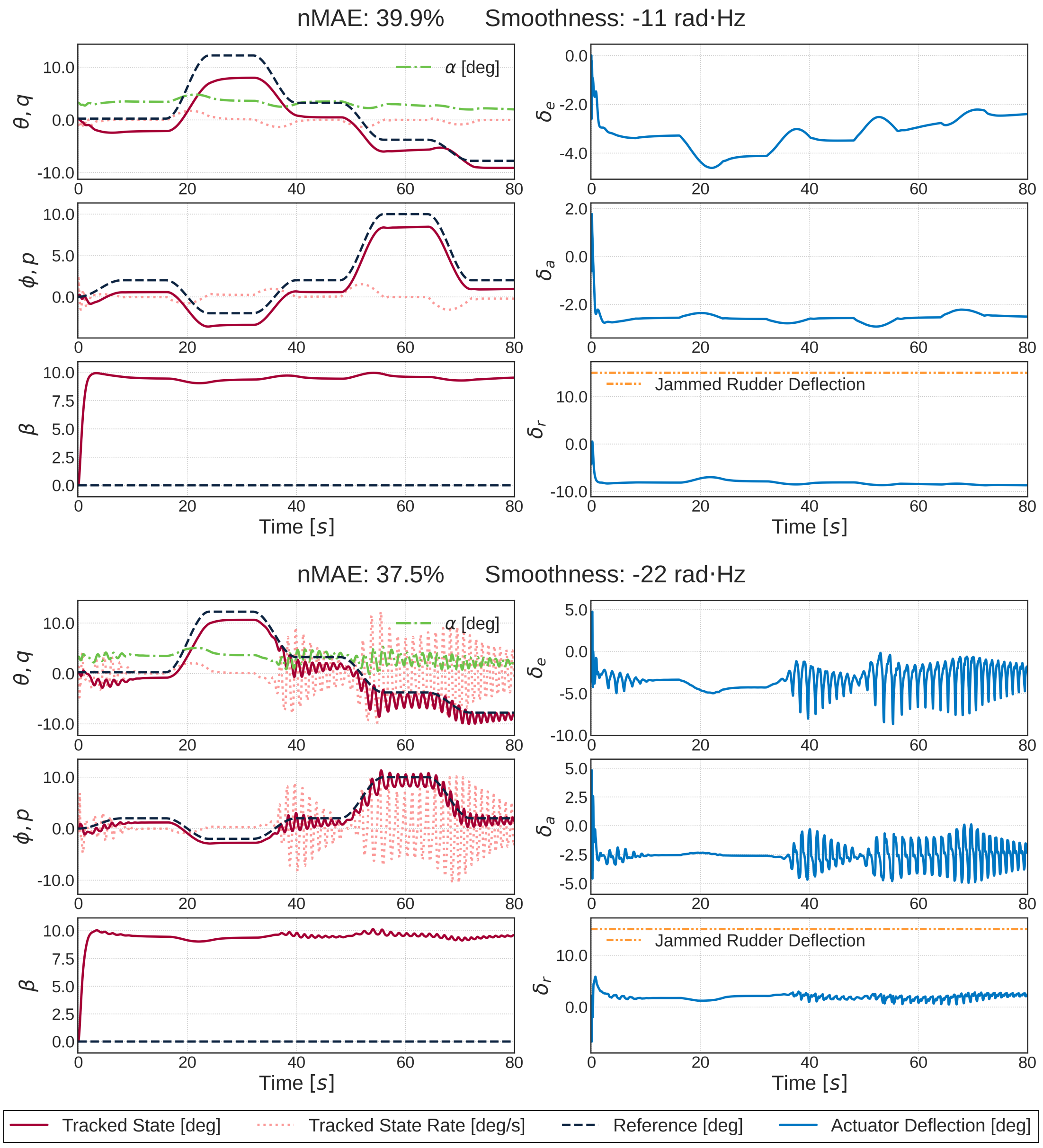


**Fig. 6 Evaluation time-traces of the SERL(50) champion (top) and TD3 (bottom) policy with their corresponding nMAE and $Sm$ metric for the jammed rudder fault (F6). The real rudder deflection is shown as a separate interrupted line.**

and translates into a higher average tracking error and standard deviation. The SERL(50) champion sees an increase of 60.1% which is lower than the one reported by [49] for a less difficult task. In this case, the RL-only agent proves more robust as its nMAE increases by 31.4%, the least among all three.

Lastly, adding biased noise does not significantly influence tracking performance. Moreover, the policies can reject a 15 ft/s vertical wind gust. The SERL champion remains the same for the nominal case, hinting that, in contrast to the faulty scenarios, genetic diversity is not necessary to achieve robustness to this atmospheric disturbance.

Overall, both SERL agents achieve relatively small changes in nMAE, comparable to the ones previously reported in the literature. Thus, they remain robust to the change in initial flight conditions and external disturbances. The TD3 agent is less capable to reproduce its nominal performance once the conditions change. Nevertheless, the low dynamic pressure case remains the hardest for all controllers, pointing towards the need for more development.

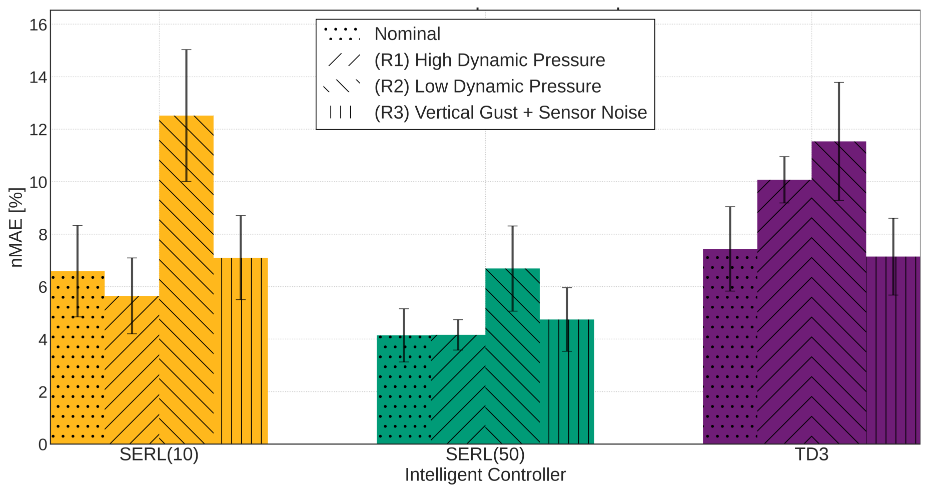


**Fig. 7 Average nMAE and standard deviation for SERL(10), SERL(50) and TD3 controller. Hatching corresponds to the evaluation case.**

## D. Control Policy Smoothness

In the time traces shown by the previous sub-sections, a key difference surfaces when comparing the SERL and the RL-only actors. The control policy smoothness is always lower for the TD3-trained controllers by at least one order of magnitude. This is not only visible via the printed $Sm$ metric but also from the high-frequency components present in the actuating signals.

The possibility to improve the population smoothness was investigated by doing an ablation study on fitness and objective function. As described in Section III, one has two options to optimise for smooth control policy, namely using CAPS and adding the $Sm$ term to the fitness function. Spectral components of the tracked reference signals are also present in the actuating signal but simply subtracting their effect from the computed $Sm$ value is not readily possible. Thus, using the same set of references for all agents enabled a valid comparison.

The scatter plot from Figure 8 compares the episodic return of the champion with the smoothness $Sm$ metric averaged over the population. It shows the three intelligent agents (colour coded) with and without CAPS regularisation and smoothness term in the fitness function (depicted via a different shape and outline). For SERL(50), the plots do not show the cases that used CAPS as the overall impact of TD3 is insignificant.

Increasing the population size affects the performance and control smoothness at different scales. The SERL(50) agent obtains a reward comparable with the CAPS-regularised TD3 and SERL(10) but it maintains a higher and almost constant policy smoothness at $Sm = -5 \pm 1$ rad · Hz. With the highest performance at $R = -78.9 \pm 21.3$, TD3 suffers from the high-frequency action noise already visible in the time traces, with an $Sm = -1500 \pm 123$ rad · Hz. The effect of both CAPS and smooth fitness are visible, significantly shifting the $Sm$ metric towards the right.

As expected, adding the $Sm$ term to the fitness function improves the achieved control smoothness but it also reduces the selective pressure for performance. Furthermore, when the TD3 actor is not regularised, it does not pass the evolutionary selection step. While SERL(50) overcomes this issue, the small population of SERL(10) does not and its top performance deteriorates. In its case, the optimisation is driven by the injected TD3 parameters, so not selecting causes the returns to collapse to $R \approx -380$. The problem is only partially mitigated by using CAPS.

### *1. Smoothness distributions*

The standard deviation provides an estimation of the spread of a probability distribution function. When the distribution is not symmetric, the standard deviation can become misleading. To Investigate such limitations, the histograms from Fig. 9 depict the distributions over the population of the tracking error and the policy control smoothness. The statistics were taken for the SERL(50) actors evaluated on the nominal case.

The $Sm$ distribution shows large left-sided skewness. At the end of the training, the majority of policies within the population are smooth while a few outliers remain noisy. As the TD3 policy is known to produce noisier actions, one can expect the outliers to trace back to TD3-trained weights injected during synchronisation. As high smoothness does not have to correlate with accurate tracking, these actors have so far survived the selection process. In spite of that, the

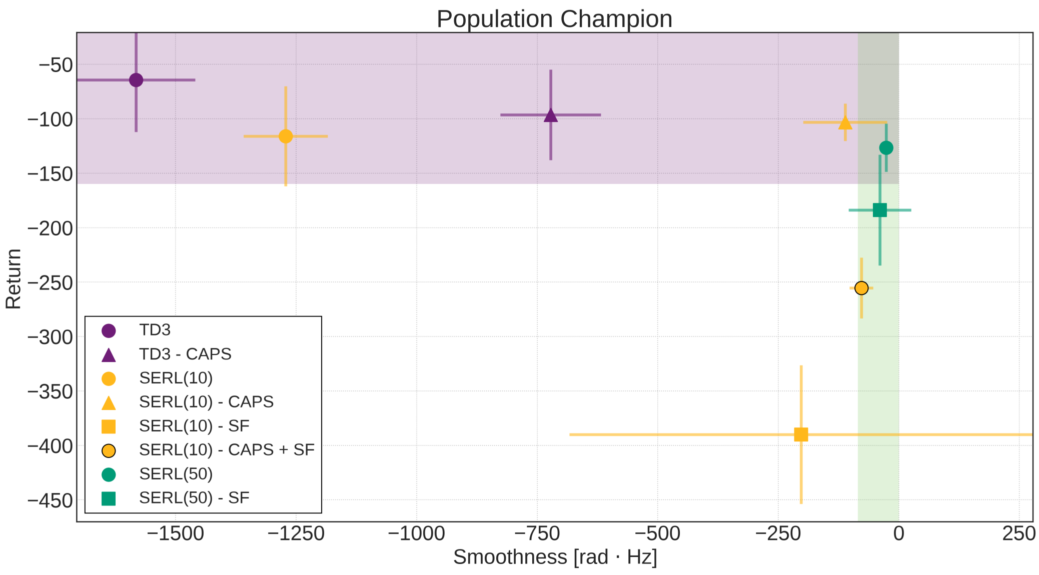


**Fig. 8 Champion return versus average smoothness for three agents trained with and without CAPS regularisation and a smoothness optimisation term.**

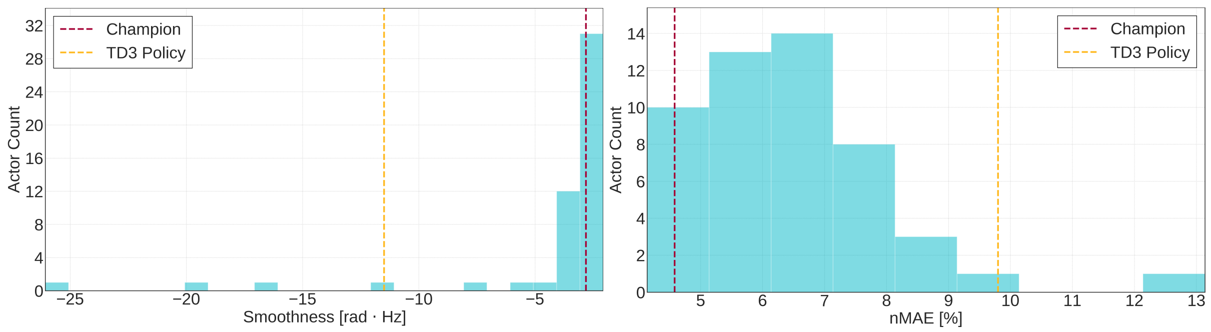


**Fig. 9 Distribution density function of the policy smoothness (left) and fitness (right) over the evolved population at the end of training.**

champion policy is among the least noisy individuals, hinting that the SERL(50) algorithm maintains smoothness as an indirect form of elitism.

The left-hand side histogram shows a higher degree of symmetry in the nMAE distribution over the population. During training, the progress of the champion is followed while the majority of genetic actors are still exploring. When the champion has converged, there is no guarantee of others' convergence, but, by not overfitting the training task, they have preserved the accumulated diversity.

### 2. *Causes*

To explain the control noise phenomena, the current work presents the following three factors which have been inferred from the aforementioned empirical ablation study and the surveyed literature:

1) *Using (deep) neural networks as optimal policy function approximators*: Multi-layered NNs have a significant number of degrees of freedom (DOF), justifying their popularity in learning complex non-linear dynamics [6]. Depending on how and for how long they are trained, NNs can learn any frequency component within the training signal. Without being biased towards them, this also includes the high-frequency components. Nevertheless, the frequency responses and Fourier analysis of neural networks are not yet a well-researched field.
2) *Exploiting environment dynamics*: The simulation environment does not penalise high-frequency components in the actuating signals as they are damped out by the aircraft dynamics. Hence, they are not present in the

tracked states. At the same time, a real-life system with unideal actuators and friction would further reduce the high-frequency comments. By not modelling such phenomena or penalising the noise via the reward or fitness signals, the agents could learn it. In turn, this would eventually aggravate the already-present gap between simulation and reality.

3) *Exploratory strategy*: As an off-policy learning framework, TD3 explores during training by adding zero-mean Gaussian noise to the selected actions. Thus, the sampled transitions used by the SGD updates are already corrupted by noise. The NNs could learn to overfit these noisy dynamics. Empirical evidence suggests that the absence or damping of additive state-action exploratory noise can benefit the action smoothness. The analysis presented by [13], has empirically shown that on-policy methods, e.g. PPO, or stochastic policy algorithms, such as SAC, are less prone to train noisy control policies.

Whereas the first two factors are not directly generating action noise, they enable it. Thus, one can argue that the developed TD3 controller learns less smooth policies in comparison to the SERL as a consequence of combining the three aspects.

Previously, other works have circumvented the noise problem by using incremental control [11, 53] or low-pass filtering of the action signal computed by the NN. These methods directly target the second factor. Similarly, using simpler control architectures, such as a PID, or regularising the neural network, as done by CAPS, also reduces the actuating noise since it diminishes the impact of the first factor.

Consequently, another hypothesis states that the neural evolution performed by the GA mitigates the impact of the third factor. While state-action noise is required for off-policy exploration of the TD3 agent, SERL can explore the policy parameter space, learning without generating noisy transitions. In parallel, the safe mutation operator biases exploratory learning towards the non-critical regions of the state space. The resulting behaviours propagate through the genetic population via distillation crossover which limits the development of aggressive behaviours via the $L_2$-norm. Thus, SERL can train in the large-population configuration without deteriorating the average control smoothness.

#### 3. The smoothness-reward trade-off

Understating the trade-off between performance and action smoothness is relevant for bridging the gap between simulation training and the deployment of hardware. Arguably, online low-pass filtering of the control commands can mitigate the effects of noise. Despite this, they can have unpredicted consequences on the neural network controllers due to their highly non-linear structure [13]. Thus, tuning the filters for multiple operating conditions might require real-life testing [13]. Similarly, changing the training task to incremental control achieves a filtering effect. Unfortunately, it also inherently increases the dimensionality of the learning space, hindering convergence and increasing the rate of failed trials [11].

Looking back at Fig. 8, two distinct regions emerge. Sometimes, a certain degree of action noise could be tolerated. Applications such as drone racing prioritise short-term performance and the small actuators can be replaced after the competition in case of damage to the high-frequency commands. These vehicles might therefore benefit more from controllers trained within the top-left configurations. In contrast, when the controllers are deployed on aircraft carrying sensitive payloads, such as scientific instruments or passengers, performance might be sacrificed to ensure that smooth commands are followed. For these applications is more appropriate to train agents in the right-hand region.

In both cases, the designer should have the opportunity to decide, but using SERL(50) achieved the best of both worlds at the cost of proportionally longer training time.

## V. Conclusion

This work combined two bio-inspired frameworks, DRL and EA, culminating in SERL, a safety-informed Evolutionary Reinforcement Learning method that trains controllers on a non-linear fixed-wing aircraft model to provide optimal attitude tracking. It uses an actor-critic RL structure with GA mechanisms. The TD3 algorithm improves sample efficiency and stability of the learning. The GA population evolves through a novelty-seeking crossover operator and safety-informed mutation that uses high-level domain knowledge about the flight envelope. With them, SERL obtains parameter-space exploration adding to the off-policy state space exploration of TD3.

The current work empirically proved that, by adjusting the population size and then selecting the adequate training time, SERL can balance performance and control smoothness. Moreover, off-policy exploration using unregularised multi-layer neural networks was shown to aggravate the action noise phenomenon in simulation environments which only prioritise tracking performance.

Training a large controller population required proportionally more samples but achieved significantly higher fault tolerance than the TD3 agent in five out of six evaluation cases. It also remained robust to flight conditions unencountered during training and external disturbances in the presence of realistic observation noise. This demonstrated the benefit of evolving a diverse population of controllers through distillation crossover and safety-informed mutations. Nevertheless, when system faults collapse the offline-obtained diversity, the TD3 remains marginally superior.

### A. Significance of Contributions

With the increasing popularity of artificial intelligence, research in control systems has the challenge and opportunity to adapt and synchronise its state-of-the-art algorithms with novel bio-inspired intelligence methods.

Directly balancing the performance-smoothness trade-off and benefiting from its proven robustness, the offline trained SERL agents can be more flexibly deployed on real-hardware applications. This represents a step towards including artificial intelligence algorithms within autonomous fault-tolerant controllers, making them a safety net applicable to safety-critical systems in general and to aircraft in particular.

### B. Recommendations

*Improve sample-efficiency:* The hybrid methods suffer from poor sample efficiency. Ideally, such issues could be targeted by improving the sample efficiency of either or both learning frameworks. On one side, the work of [47] shows the opportunity to increase the training stability and efficiency of the RL agent by involving distributional critics. This could potentially benefit large population searches which hinder the DRL convergence rate. On the other side, novel evolutionary strategy algorithms, such as Covariance Matrix Adaptation [26], can replace the GA loop. For these extensions, the control policy smoothness and the possibility to add it to the fitness function shall be closely investigated.

*Online actor selection:* Each actor has been trained to control the system. To select which one becomes the actual behavioural policy, the decision problem can be formulated as short-term receding horizon optimisation. Online system identification tools can be used to model the local dynamics and estimate each actor's performance (or another desired metric). Then, a new champion policy is selected and the behavioural policy is shifted towards it by either a hard or soft update.

*Flight-test Validation:* Proving robustness to changes in flight conditions, system and sensor dynamics and external disturbances, the work made valuable steps towards showing that SERL can provide attitude control on real systems. Despite this, validating its use would require flight testing. Generally, the problem of validating neural networks as controller policies is cumbersome due to their black-box nature, which lacks explainability.

*6-DOF Flight Control:* This work only targeted the attitude control task and trained in one flight condition. Previous works have shown the opportunity to train 6-DOF controllers in a hierarchical manner. Achieving smooth control via direct actuator commands and benefiting from episodic evolutionary learning, the present work opens a possibility to circumvent the curse of dimensionality of 6-DOF control. Moreover, training such a controller on multiple points sampled within the flight envelope can increase its performance in the low dynamic pressure regime.

## References

[1] "Safety Report," Tech. rep., International Civil Aviation Organization (ICAO), 2020. URL `https://www.icao.int/safety/iStars/Pages/Accident-Statistics.aspx`.

[2] Lu, P., van Kampen, E.-J., De Visser, C., and Chu, Q., "Aircraft fault-tolerant trajectory control using Incremental Nonlinear Dynamic Inversion," *Control Engineering Practice*, Vol. 57, 2016, pp. 126–141.

[3] Edwards, C., Lombaerts, T., Smaili, H., et al., "Fault tolerant flight control," *Lecture Notes in Control and Information Sciences*, Vol. 399, 2010, pp. 1–560.

[4] Lavretsky, E., and Wise, K. A., "Robust adaptive control," *Robust and adaptive control*, Springer, 2013, pp. 317–353.

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

[6] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al., "Human-level control through deep reinforcement learning," *nature*, Vol. 518, No. 7540, 2015, pp. 529–533. https://doi.org/https://doi-org.tudelft.idm.oclc.org/10.1038/nature14236.

[8] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., "Mastering the game of Go with deep neural networks and tree search," *nature*, Vol. 529, No. 7587, 2016, pp. 484–489.

[9] Tsourdos, A., Permana, I. A. D., Budiarti, D. H., Shin, H.-S., and Lee, C.-H., "Developing flight control policy using deep deterministic policy gradient," *2019 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*, IEEE, 2019, pp. 1–7.

[10] Choi, M., Filter, M., Alcedo, K., Walker, T. T., Rosenbluth, D., and Ide, J. S., "Soft Actor-Critic with Inhibitory Networks for Retraining UAV Controllers Faster," *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2022, pp. 1561–1570.

[11] Dally, K., and Van Kampen, E.-J., "Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control," *AIAA SCITECH 2022 Forum*, 2022, p. 2078.

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

[13] Mysore, S., Mabsout, B., Mancuso, R., and Saenko, K., "Regularizing action policies for smooth control with reinforcement learning," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 1810–1816.

[14] Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B., "Robots that can adapt like animals," *Nature*, Vol. 521, No. 7553, 2015, pp. 503–507.

[15] Doncieux, S., Bredeche, N., Mouret, J.-B., and Eiben, A. E. G., "Evolutionary Robotics: What, Why, and Where to," *Frontiers in Robotics and AI*, Vol. 2, 2015. https://doi.org/10.3389/frobt.2015.00004.

[16] Dianati, M., Song, I., and Treiber, M., "An introduction to genetic algorithms and evolution strategies," Tech. rep., University of Waterloo, Waterloo, Ontario, Canada, 2002.

[17] Fontaine, M. C., Togelius, J., Nikolaidis, S., and Hoover, A. K., "Covariance matrix adaptation for the rapid illumination of behavior space," *Proceedings of the 2020 genetic and evolutionary computation conference*, 2020, pp. 94–102.

[18] Papavasileiou, E., Cornelis, J., and Jansen, B., "A Systematic Literature Review of the Successors of "NeuroEvolution of Augmenting Topologies"," *Evolutionary Computation*, Vol. 29, No. 1, 2021, pp. 1–73.

[19] Stanley, K. O., and Miikkulainen, R., "Evolving neural networks through augmenting topologies," *Evolutionary computation*, Vol. 10, No. 2, 2002, pp. 99–127.

[20] Lim, B., Grillotti, L., Bernasconi, L., and Cully, A., "Dynamics-aware quality-diversity for efficient learning of skill repertoires," *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 5360–5366.

[21] Kaushik, R., Desreumaux, P., and Mouret, J.-B., "Adaptive prior selection for repertoire-based online adaptation in robotics," *Frontiers in Robotics and AI*, 2020, p. 151.

[22] Chatzilygeroudis, K., Vassiliades, V., and Mouret, J.-B., "Reset-free trial-and-error learning for robot damage recovery," *Robotics and Autonomous Systems*, Vol. 100, 2018, pp. 236–250.

[23] Franke, J. K., Köhler, G., Biedenkapp, A., and Hutter, F., "Sample-efficient automated deep reinforcement learning," *arXiv preprint arXiv:2009.01555*, 2020.

[24] Khadka, S., and Tumer, K., "Evolution-guided policy gradient in reinforcement learning," *Advances in Neural Information Processing Systems*, Vol. 31, 2018.

[25] Conti, E., Madhavan, V., Petroski Such, F., Lehman, J., Stanley, K., and Clune, J., "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," *Advances in neural information processing systems*, Vol. 31, 2018.

[26] Hansen, N., "The CMA evolution strategy: a comparing review," *Towards a new evolutionary computation*, 2006, pp. 75–102.

[27] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I., "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[28] Bodnar, C., Day, B., and Lió, P., "Proximal distilled evolutionary reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, 2020, pp. 3283–3290.

[29] Khadka, S., Majumdar, S., Nassar, T., Dwiel, Z., Tumer, E., Miret, S., Liu, Y., and Tumer, K., "Collaborative evolutionary reinforcement learning," *International conference on machine learning*, PMLR, 2019, pp. 3341–3350.

[30] Sigaud, O., "Combining Evolution and Deep Reinforcement Learning for Policy Search: a Survey," *arXiv preprint arXiv:2203.14009*, 2022.

[31] Pourchot, A., and Sigaud, O., "CEM-RL: Combining evolutionary and gradient-based methods for policy search," *arXiv preprint arXiv:1810.01222*, 2018.

[32] Marchesini, E., Corsi, D., and Farinelli, A., "Exploring safer behaviors for deep reinforcement learning," *AAAI Conference on Artificial Intelligence*, 2021.

[33] Bertsekas, D., *Reinforcement learning and optimal control*, Athena Scientific, 2019.

[34] Thrun, S., and Schwartz, A., "Issues in using function approximation for reinforcement learning," *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, Vol. 6, 1993.

[35] Fujimoto, S., Hoof, H., and Meger, D., "Addressing function approximation error in actor-critic methods," *International conference on machine learning*, PMLR, 2018, pp. 1587–1596.

[36] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[37] Polyak, B. T., "Some methods of speeding up the convergence of iteration methods," *Ussr computational mathematics and mathematical physics*, Vol. 4, No. 5, 1964, pp. 1–17.

[38] Zhang, B.-T., Muhlenbein, H., et al., "Evolving optimal neural networks using genetic algorithms with Occam's razor," *Complex systems*, Vol. 7, No. 3, 1993, pp. 199–220.

[39] Koza, J. R., and Poli, R., "Genetic programming," *Search methodologies*, Springer, 2005, pp. 127–164.

[40] Wiering, M. A., and Van Otterlo, M., "Reinforcement learning: State-of-the-art," *Adaptation, learning, and optimization*, Vol. 12, No. 3, 2012, p. 729.

[41] Pugh, J. K., Soros, L. B., and Stanley, K. O., "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, Vol. 3, 2016, p. 40.

[42] Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M., "Parameter space noise for exploration," *arXiv preprint arXiv:1706.01905*, 2017.

[43] Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R., "Policy distillation," *arXiv preprint arXiv:1511.06295*, 2015.

[44] Linden, V. D., "DASMAT-Delft University aircraft simulation model and analysis tool: A Matlab/Simulink environment for flight dynamics and control analysis," *Series 03: Control and Simulation 03*, 1998. URL http://resolver.tudelft.nl/uuid:9bfb1f68-2f7c-4f32-91b4-79297d295f84.

[45] Van den Hoek, M., de Visser, C., and Pool, D., "Identification of a Cessna Citation II model based on flight test data," *Advances in Aerospace Guidance, Navigation and Control*, Springer, 2018, pp. 259–277. URL http://resolver.tudelft.nl/uuid:c0a57513-38b7-4d3a-898c-fa57c3e7ac2e.

[46] 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. URL http://resolver.tudelft.nl/uuid:b37cefbf-e353-43cf-8d9d-98f2653216c6.

[47] Seres, P., and Van Kampen, E.-J., "Distributional Reinforcement Learning for Flight Control: A risk-sensitive approach to aircraft attitude control using Distributional RL," Tech. rep., Delft University of Technology, 2022. URL http://resolver.tudelft.nl/uuid:6cd3efd1-b755-4b04-8b9b-93f9dabb6108.

[48] EASA, "Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes (CS-25)," Tech. rep., European Union, June 2021. URL https://www.easa.europa.eu/en/document-library/certification-specifications/cs-25-amendment-27.

[49] Teirlinck, C., "Reinforcement Learning for Flight Control: Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance," Tech. rep., Delft University of Technology, 2022. URL http://resolver.tudelft.nl/uuid:dae2fdae-50a5-4941-a49f-41c25bea8a85.

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

[51] Grondman, F., Looye, G., Kuchar, R. O., Chu, Q. P., and Van Kampen, E.-J., "Design and flight testing of incremental nonlinear dynamic inversion-based control laws for a passenger aircraft," *2018 AIAA Guidance, Navigation, and Control Conference*, 2018, p. 0385.

[52] Moorhouse, D. J., and Woodcock, R. J., "Background information and user guide for MIL-F-8785C, military specification-flying qualities of piloted airplanes," Tech. rep., Air Force Wright Aeronautical Labs Wright-Patterson AFB OH, 1982.

[53] Völker, W., "Reinforcement Learning for Flight Control of the Flying V," Tech. rep., Delft University of Technology, July 2022. URL http://resolver.tudelft.nl/uuid:a6b645d2-8d47-44d3-a4ad-1d5a6024f13f.

# Part II

## Preliminary Analysis

*This part has been assessed for the course AE4020 Literature Study.

# 3 Literature Review

This chapter identifies from the field literature a hybrid algorithm that can optimise policies and verify on a toy-problem control task their robustness against relevant fault cases.

It starts by presenting on a high-level the current approaches to fault-tolerant flight control, introducing the concepts of robust and adaptive control. Section 3.1 focuses on the main faults that such flight controllers have to deal with, introducing the idea of using bio-inspired solutions to overcome some of the challenges of intelligent control. Then, Section 3.2 introduces deep reinforcement learning as the first bio-inspired class. The chapter will look at the basic principles, identify the state-of-the-art and comment on the advantages and limitations that it poses. Section 3.3 will present the second class of methods, evolutionary algorithms, and will focus on its applicability to optimise neural networks for fault-tolerant control problems. Combining both frameworks, the novel field of hybrid learning methods emerges in Section 3.4.

## 3.1. Intelligent Fault-tolerant Flight Control

In general, a fault-tolerant controller would aim to remain performant according to a specific metric) when controlling systems the presence of faults. Its architecture is designed to overcome failures of different aircraft systems, errors in the signals from sensors or subsequent unpredicted phenomena. All of these are commonly referred to as faults. Aiming to answer the first research question, namely **Met-1**, this chapter identifies the most prevalent fault cases and provides an overview of the current approaches that deal with them. Later sections will detail the promising concepts introduced by this chapter.

Whereas Section 3.1.1 introduces the modern approaches in robust and adaptive control and then Section 3.1.2 compiles a list of faulty scenarios that a flight controller has to withstand.

### 3.1.1. Modern Robust and Adaptive Approaches

Fault tolerance is correlated to the frameworks of robust and adaptive control techniques. Therefore, one should first look into what these two umbrella terms refer to. According to Lavretsky and Wise [4], a *robust* controller is expected to perform 'relatively' well under the assumption of worst-case conditions. In contrast, an *adaptive* controller would try to perform an online estimation of the process uncertainty and then produce a control input to anticipate, overcome, or minimise the undesirable deviations from the prescribed behaviour [4].

A successful approach to dealing with faults involves active adaptive flight control systems (FCS). Such a fault-tolerant controller system (FTC) compares signals from both control and monitoring channels to detect failures. This algorithm must be reliable enough to provide tolerance to sensor inaccuracies and modelling errors to prevent false alarms, but sensible enough to detect failures on time. Figure 3.1, adapted from [3], shows an example of such a flight control system. Here, the states of the aircraft and the outputs of its subsystems are compared to reference values within a Fault Detection & Diagnosis (FDD) block. The reference signals are obtained by feeding the recorded data into a model of the aircraft dynamics which can be identified beforehand (offline) or during the flight (online). Once a fault is identified, the diagnoser sends a command to the control 'Reconfiguration mechanism', a system that updates the policy of the controller according to the identified fault case. In many modern aircraft, such systems are usually implemented in the form of gain schedulers, adapting the gains of linear PID controllers [3].

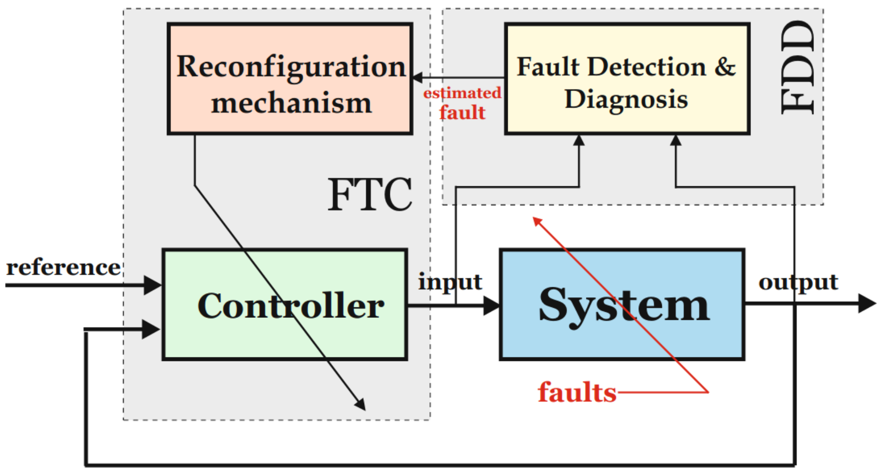


**Figure 3.1:** Generic fault-tolerant control scheme used within the FCS [3].

Besides controllers that can be adapted, modern systems also incorporate detailed procedures for safety checks and hardware redundancy. This work will only focus on the software solutions which are currently the subject of competitive industrial and academic research.

Nowadays, the state-of-the-art in field fault-tolerant flight control revolves around intelligent adaptive methods. In addition to their adaptive properties, such controllers can be constructed to ‘learn’, or to ‘intelligently remember’ useful observed information. In most general terms, learning refers to remembering and/or recognising certain patterns in the observations and acting based on prior knowledge [4]. One can split them according to multiple taxonomies such as model-free or model-based, online and offline learning. Furthermore, hybrid methods are developing, for example, consider an offline pre-trained model which is then updated online. The detailed benefits of both sides from the above list of alternatives will unveil themselves throughout the next chapters. This section serves as an overview of the current field’s efforts, hopes and goals.

Artificial Neural Networks (ANNs) have seen a massive spike in popularity due to their success within the deep learning framework [10]. Despite this, the idea of training a non-linear regression model, such as a Neural Network (NN) is by no means new. As part of NASA’s Intelligent Flight Control System (IFCS) program for F-15 program [47], pre-trained and online learning neural networks were flight tested on the NASA IFCS F-15 test bench. The experiments validated the ability of the neural networks to predict the stability parameters of the aircraft used for control.

More recently, techniques such as gain scheduling fuzzy logic control [48] have steadily developed into popular approaches for controlling systems with faults. Historically, the standard methods try to first identify the type of fault (the ’Diagnoser’ block from Figure 3.1 is responsible for this) and select a control policy according to the specific fault. the underlying requirement is that the controller has to be trained on as many fault cases as possible. When a new situation is encountered, the gain scheduler or fuzzy logic operator can then interpolate between the known controllers. For example, [48] presents a hybrid architecture that uses an augmented Takagi-Sageno fuzzy controller in a reinforcement learning framework to find control policies for partially unknown systems in presence of actuator faults. The method can also extrapolate to combinations of known faults. It is able to perform in real-time but the recorded faults have to fall within pre-defined bounds, thus greatly reducing its generalisability.

Both the aerospace and robotics literature seem to agree that the task of identifying the fault is not a well-posed problem. Mathematically, such architectures cannot distinguish between faults in the systems and modelling errors as both would cause mismatches between signals [21, 3]. Moreover, faults such as icing or biased sensors are particularly difficult to detect or isolate [18]. In flight control, non-linear methods such as Incremental Non-linear Dynamic Inversion (INDI) or incremental backstepping (IBS) alleviate the need for the diagnosis step. The incremental part is responsible for the online identification of linear and time-varying models of the aircraft dynamics, reducing the sensitivity to mismatches in the model [2, 8].

Reinforcement learning (RL) is a popular learning-based approach to solving complex tasks. It has been applied to flight control as Approximate Dynamic Programming. This framework is still reliant on identifying a dynamic model online, but it demonstrated robustness in dealing with faults [6]. This proved successful in controlling highly complex coupled systems such as the F-16 fighter jet aircraft [7] due to its higher approximation power than other previously described non-linear control methods.

Recently, research has started focusing on offline methods which can significantly improve performance by training longer in a simulation environment. Bio-inspired methods such as deep reinforcement learning (DRL) remove the need for identifying the fault, moving towards model-free adaptive policies [19, 18, 16]. The intelligent controller trains by experiencing many simulated interactions between the system and the environment. After training is completed offline, the controller can follow the converged policies, hopefully dealing with the disturbances, faults and other unforeseen phenomena. Moreover, the DRL framework inherits the advantages of modern deep learning techniques [10]. Essentially, they provide a relatively-sample efficient way to train non-linear functions to approximate complex dynamics [5]. Leveraging longer training times, DRL obtains state-of-the-art performance in tasks considered difficult for more traditional control systems [18, 19].

To better understand the reasoning behind model-free intelligent control, one shall look back at what fault tolerance would require. An agent, be it a human pilot or a specific form of high or low-level AI within the auto-pilot, has been trained to control the system in a nominal scenario. Once some damage is inflicted, the dynamics of the system change. Does it still show any resemblance to the nominal system? Yes but in a limited way. Now the agent has to adjust or adapt its behaviour to the new system. Unfortunately, the time required to re-learn an entirely novel control policy is much longer than the one available. Thus, to avoid imminent disaster, the controller shall efficiently use a sub-set of its prior knowledge and tweak its control policy. Day-to-day empirical evidence has shown that humans and animals are creatively discovering compensatory behaviours without being limited to a set of rigorously predefined rules or assumptions about how damage may occur and how to compensate for each type of fault [3].

The AI-based controller can also benefit from a similar type of curiosity. In general terms, the machine has to be able to deal with unforeseen circumstances. Within the machine learning community, this is widely referred to as *generalisability* [10]. A generalisable learning method would translate into a control algorithm capable to react once a new set of state-action pairs is fed through or unforeseen phenomena appear. Ideally, its performance shall at most gracefully degrade in **out-of-distribution conditions**.

But implementations of intelligent controllers have their own challenges to be tackled. Section 3.2.4 will detail how DRL is implemented in flight control and Section 3.3.3 will present a few approaches to using evolutionary algorithms to train a diverse repertoire of policies to adapt to faults.

### 3.1.2. Fault Cases

Before the phase of controller design, one shall first aim to isolate the corresponding faulty conditions that the adaptive and/or robust controller has to be able to cope with.

In [3], a detailed historical analysis of air crashes has been conducted. The most critical failure cases are identified as the probable scenarios that would be in the most severe accidents. The author considers the rudder hardover, engine separation and total loss of hydraulics as the most critical failure cases. As they are considered the most severe, these scenarios can be considered the extreme corners of a failure envelope. Within the span of such an envelope, there are other cases that a fault-tolerant controller should be able to deal with. The following list has been compiled after consulting multiple works that consider aspects of designing fault-tolerant controllers:

1. Changed aerodynamic parameters (mainly lift and drag coefficients) [18]. For example, this can be caused by icing which increases the skin friction drag and reduces the maximum lift attainable and the stall angle by downshifting the lift curve [49].
2. Sudden change of centre of gravity (CG) location. Such fault case affects larger aircraft less than smaller business jets or small unmanned aerial vehicles (UAVs).
3. Reduce control surface effectiveness. For a standard fixed-wing aircraft this implies that either the rudder, the elevator or the ailerons are less effective due to, for example, damage or mechanical friction [2].

4. Faulty sensor recordings. Bias or increased noise in the sensor data can be highly detrimental to state estimation and control [18].
5. Structural damage of a control surface which causes total loss of control for that actuator [18, 2].
6. Lack of control after an engine burst. The controller has to counteract large asymmetric moments caused by the thrust differential [3].
7. *Undetected oscillatory failure at a frequency critical to the structure [50]. Such modes of vibration are caused by aeroelastic effects such as flutter or by pilot-induced oscillations, a phenomenon known from manual control. The fault-tolerant controller would aims to prevent the onset of such oscillations, reducing the risk for future damage.

Lastly, multiple faults can occur, essentially combining two or even more categories from the list above. Icing can affect the aerodynamic parameters of the aircraft but it can also reduce the effectiveness of the actuators. Similarly, debris from an engine burst can inflict structural damage on the tail surfaces, as happened during the United Airlines Flight UA232 accident [51]. The combination of multiple failure cases would not only increase the severity but also make the control more difficult. One can argue that, in such cases, preparing for each failure scenario would result in reduced control effectiveness [3].

## 3.2. Deep Reinforcement Learning

This chapter presents the reinforcement learning methods, starting from its mathematical background in dynamic programming and moving towards their applications to state-of-the-art intelligent flight control. By carefully considering their strengths and limitations, they will later become building blocks for the hybrid framework developed throughout this research. Thus, this chapter contributes to the body of research work by answering sub-question **Met-2.b**.

The chapter starts by providing a formal theoretical background in Section 3.2.1 and it moves towards the more advanced topic of learning with function approximators in Section 3.2.2. Then, three state-of-the-art algorithms are selected and described in Section 3.2.3. Section 3.2.4 will shift the perspective toward the applicability of DRL to flight control problems. The chapter concludes with Section 3.2.5 by reflecting on the advantages and disadvantages of reinforcement learning.

### 3.2.1. Background on Reinforcement Learning

Reinforcement learning (RL) is a machine learning framework directly inspired by the learning processes of living beings. In the most general sense, it allows a programmed *agent* to interact with its *environment* and learn 'what to do — how to map situations to actions — so as to maximise a numerical reward signal' [5]. The learner neither knows nor is told beforehand what will happen when a certain action is taken. By recursively iterating on its behaviour, it can eventually learn the correct actions that lead to its desired goal.

Another perspective on reinforcement learning had been proposed by Bertsekas and Tsitsiklis and it has roots in the field of operational research [52]. In comparison to the framework of Sutton and Barto, the differences are subtle and mainly come from the authors' backgrounds. Sutton and Barto aim to learn by maximising *rewards* and talk about *value functions* [5] whereas Bertsekas and Tsitsiklis talk about *cost-to-go-functions*, always moving towards the minimal cost [52]. This work will stick to the perspective of Sutton and Barto as it is currently more established in the field of RL.

#### Basic concepts

Any RL task can be summarised as the trial of an agent to move from its initial state to the final state while maximising a reward $r$ provided by the environment. In the case of aircraft flight control, the task is continuous so the final state or termination criterion is usually specifically defined for each task [53].

At each time-step, it selects an action $a_t$ to move from its current state $s$ to the future state $s'$. Once the terminal state has been reached, the episode is completed and the sum of all rewards can be reported. Figure 3.2 depicts the agent-environment interaction.

For the current problem, the learning agent is the flight-control control system and the environment corresponds to the aircraft and the volume of air through which it flies.

#### Markov Decision Process

All approaches of RL place at their core the mathematical assumption of a fully-observable Markov Decision Process (MDP). To formalise the MDP, the literature defines the following functions and variables:

- $\mathcal{S}$ is the set of all valid states
- $\mathcal{A}$ is the set of all valid actions
- $\mathcal{R}: S \times A \times S \rightarrow \mathbb{R}$ is the reward function, with $r_t = \mathcal{R}\left(s_t, a_t, s_{t+1}\right)$
- $\mathcal{P}: S \times A \rightarrow S$ is the state-transition (probability) function, with $\mathcal{P}\left(s' \mid s, a\right)$ being the probability of transitioning into the next state $s'$ from current state $s$ by performing action $a$. The index $t$ is usually dropped for brevity.

In optimal control, a system is considered an MDP iff it obeys the Markov property: the current transition only depends on the most recent state and action [54]. Hence, the memory of the previously observed history of transitions collapses into the current state, as shown by Equation 3.1:

$$\mathcal{P}\left\{s_{t+1}, r_{t+1} \mid s_t, a_t, \ldots, s_0, a_0\right\} = \mathcal{P}\left\{s_{t+1}, r_{t+1} \mid s_t, a_t\right\} \tag{3.1}$$

Essentially, the Markov property ensures that the agent's future can be fully explained by the present.

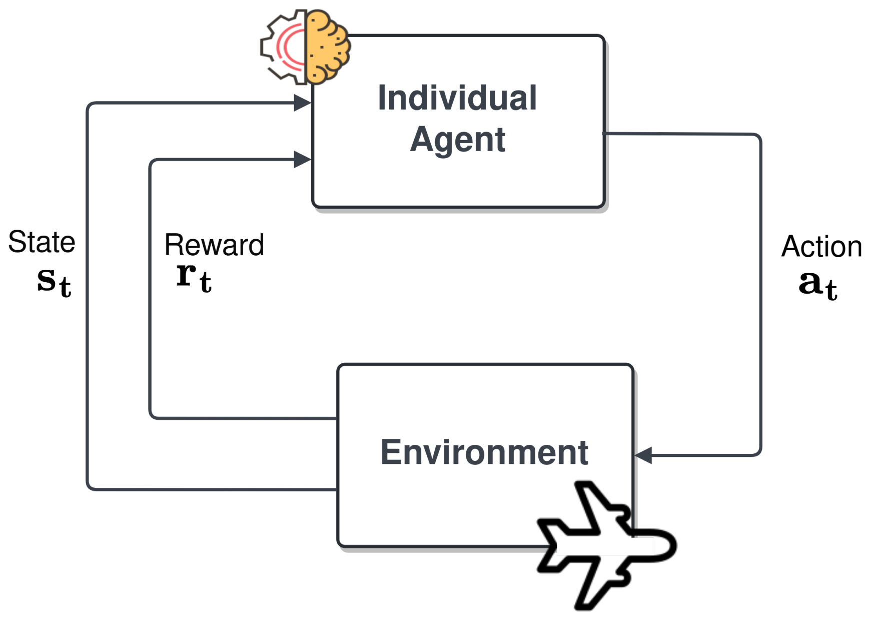


**Figure 3.2:** The agent–environment interaction in the frame of a Markov decision process, the basic exchange of information behind reinforcement learning.

There is a distinction between what the agent's observation and state. Once the agent cannot fully determine its state from the observation, it is forced to carry or learn a model of its sensors. This belongs to the more general framework of partially observable MDP (POMDP) [55]. The present RL formulation will always be assumed that the agent's state is entirely known from its observation.

**Policy and trajectory**
The policy $\mu$ can be seen as either the behaviour of the agent or the control law. In other words, it is a rule that tells the agent what actions to take at its state. It can be deterministic or stochastic, denoted by $\mu$ and $\pi$ respectively:

$$a_t = \mu(a_t) \quad \text{or} \quad a_t \sim \pi(\cdot|s_t). \tag{3.2}$$

In this section, the policy will be denoted by $\pi$ to align with the field's literature which generally assumes a stochastic policy distribution. Despite this, Section 3.2.3 will present algorithms that select actions based on deterministic policies.

Subsequently, a *trajectory* $\mathcal{T}_n$ is an ordered sequence of $n$ state-action pairs that an agent passes through while following a policy:

$$\mathcal{T}_n = (s_0, a_0, s_1, a_1, \ldots, s_{n-1}, a_{n-1}) \tag{3.3}$$

Thus, the trajectory can be seen as the motion resulting from following one or more policies.

**Reward function**
The reward function $R$ depends on the current state of the world, the action just taken, and the next state of the world:

$$r_t = \mathcal{R}(s_t, a_t, s_{t+1}) \tag{3.4}$$

As a possible simplification, the reward function is reduced to a dependence on the current state, $r_t = \mathcal{R}(s_t)$, or state-action pair $r_t = \mathcal{R}(s_t, a_t)$.

Learning is enabled by maximising the *cumulative reward* over a trajectory starting at the current timestamp $t$, referred to as the *return* $R_t$. For continuous learning, the return is the infinite-horizon sum of all rewards ever obtained by the agent, discounted by how far off in the future they are obtained:

$$R_t = \lim_{n\to\infty} (r + r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^n r_{t+n}) = \lim_{n\to\infty} \sum_{k=0}^{n} \gamma^n r_{t+k} \tag{3.5}$$

The discount factor or rate $\gamma$ from Equation 3.5 represents the confidence or interest in future events. A value close to zero disregards future rewards whereas a high discount factor favours them. The intuition is backed mathematically: the infinite-horizon sum of a series of rewards can converge if and only if the series itself is convergent. This translates to the condition that:

$$\lim_{n\to\infty} \gamma^n \cdot r_{t+n} \to 0 \tag{3.6}$$

Which always happens for $\gamma \in (0,1)$. For episodic learning, the time does not approach infinity as any episode ends after a number of steps or when the termination criterion has been reached. In practice, for long leanings episodes, the above mathematical formalism still holds.

**Value functions**
To track the expected return obtained by following a policy, the concept of state value function is introduced by Equation 3.7. It gives the expected return if you start in state $s$ and always act according to policy $\pi$:

$$V^{\pi}(s) = \mathbb{E}\left[R_t \mid s_t = s\right] \tag{3.7}$$

In a similar fashion, the state-action value function from Equation 3.8 records the expected return when the agent starts in state $s$, takes an arbitrary action and then forever after acts following policy $\pi$. It is important to note that $a_t$, the first action taken, does not have to be following policy $\pi$:

$$Q^{\pi}\left(s_t, a_t\right) = \mathbb{E}_{\pi}\left[R_t \mid s_t, a_t\right] \tag{3.8}$$

Now, by using these two functions, one can try to find the optimal policy $\pi^*$. By following this policy, the expected discounted return is maximised, yielding the optimal value functions in Equation 3.9:

$$Q^*(s,a) = max_{\pi}\mathbb{E}\left[R_t \mid s_t = s, a_t = a\right] \quad \text{and} \quad V^*(s) = max_{\pi}\mathbb{E}\left[R_t \mid s_t = s\right] \tag{3.9}$$

Lastly, the first action $a_t$ has to be selected. As a direct result of the definition of $Q^*$ is known, one can compute the optimal action, $a^*(s)$, referred to as the *greedy action*:

$$a^*(s) = argmax_a Q^*(s,a) \tag{3.10}$$

**Bellman optimality equation**
Both the state and state-action functions can be defined recursively. By re-writing Equations 3.8 and 3.7 one obtains the *Bellman optimality* equations [5, 54]:

$$Q^*(s,a) = \mathsf{E}_{s'\sim\mathcal{P}}\left[r(s,a) + \gamma \cdot \max_{a'} Q^*\left(s',a'\right)\right] \quad \text{and} \tag{3.11}$$

Where $\mathcal{P}$ denotes the markovian state-transition function introduced at the beginning of this section. Essentially, the value at the starting point is the reward you expect to get from being there, plus the value at the next state.

Richard Bellman realised that this recursive consistency enables finding optimal policies for an MDP. His insight became the fundamental building block of all RL algorithms [5].

**Learning Methods**
The literature distinguishes between three classes of methods that aim to solve the MDP: *dynamic programming* (DP), *Monte Carlo methods* and *temporal difference learning* (TD). They all apply the Bellman equation as an iterative update rule but DP and TD learn during the episode whereas Monte Carlo updates are only done at the end of it. The difference between DP and TD learning comes from their dependency on an available state-transition model with DP being *model-based* and TD considered *model-free* [5]. TD learning represents the core of all the algorithms that will be explained in this chapter.

**Temporal difference update**
This class of model-free methods learns based on bootstrapping: they use values from previous iterations

to bound the estimate of the current value. This iterative process is well-suited for control tasks. The principle is to compute the error between the target and the current estimate of the value function and increment the current estimate by this error factored by a learning rate $\alpha$. Equation 3.12 dictates the update of the value functions:

$$\begin{aligned} V(s_t) &\leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \\ Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \cdot max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right] \end{aligned} \tag{3.12}$$

In this case, the target is the optimal value functions given by Equation 3.9 and the TD-error corresponds to the expression between square brackets. The action-state value function, also known as the Q-function, is used for the *off-policy* update as it explicitly distinguishes between the greedy (optimal) action and the actual executed action. This aspect will be discussed further in the next section.

The downside of TD methods is that, in order to guarantee convergence, all state-action pairs need to be visited continually [5]. Thus, this becomes a necessary assumption for all reinforcement learning methods that will be described in later sections.

With TD as a baseline for the update rule, RL algorithms can be built to learn the optimal solution. In discrete state and action spaces, this is done with Q-Learning (using the $Q$ update) or SARSA (using the $V$ update) [5]. These methods do not learn an explicit policy but only a value function and find the action argument that maximises it. Similarly, the actor-critic algorithms described by Section 3.2.3 will also incorporate the TD update.

**Exploration and Exploitation**

When the agent always chooses the greedy action, it will find a solution, but not necessarily the optimal solution. The reason for this is that some state-action combinations have never been evaluated, as they did not correspond to the highest value at that time. Exploration is explained as the need of the agent to try out sub-optimal actions in its environment, as to increase its chances of escaping the local optima of its current policy. At the core of learning from reinforcement signals, there is a trade-off between exploration and exploitation [5].

But this trade-off goes beyond RL. One can realise that it is, by all means, present under different names in other machine learning and optimal control frameworks. Mostly relevant for this work are genetic algorithms. They explore via carefully designed mutation operators that alter the current population of solutions while exploiting ‘good’ policies by combining them via crossover. Both will be detailed in Chapter 3.3.

Last but not least, the trade-off is very much present in the daily learning endeavours of all leaving species alike. Exploration is a synonym for the curiosity that drives living creatures to find out more about their own environment. Sometimes, it causes negative effects, such as pain or time delay, with the hope that exploration will return more useful experiences in the long run. Other times, one has to accept their current situation and exploit the most out of it.

Following the previous discussion, RL distinguishes between two types of exploration. Any algorithm will be based on one or the other. The key ideas behind these two ways are seen below:

**Off-policy and on-policy learning**

- **Off-policy**: evaluate or improve a policy different from that used to generate the action(s) for the current state.
- **On-policy**: evaluate or improve the same policy currently used to make decisions.

Consequently, an off-policy setup for exploration enables learning from trajectories that are not necessarily obtained under the currently followed policy. In general, it uses the greedy action to update the Q-function while adding noise to the current executed action. During training, this exploratory noise may lead to dangerous actions with catastrophically low returns. During the subsequent evaluation step, safety is achieved by removing exploration noise. Finally, to enable off-policy learning, a finite-size memory or *replay buffer* is used to store the trajectories and re-use random samples of the different behaviour policies.

In contrast to the first class, on-policy methods such as Proximal Policy Optimisation (PPO) usually introduce a bias when used with trajectories that are not obtained under the current policy or incorporate a correction mechanism, such as importance sampling [15]. For them, exploratory behaviour is obtained by adding noise to the selected action with the $\epsilon$-greedy action being the straightforward way of doing it since early classic RL algorithms such as SARSA [5]. Alternatively, they can sample directly from a stochastic policy, as it is done by PPO [15]. Over the course of training, the policy typically becomes progressively less random, as the update rule encourages it to exploit behaviours already seen to be rewarding. Despite showing a more stable convergence, defined as the flattening of the learning curve and diminished variance of the obtained reward, this may cause the policy to get trapped in local optima [56].

Off-policy exploration is thus more sample efficient as they can achieve a higher reward faster by using (and re-using) any experience. Despite this, they can suffer from poor policy convergence [5].

Once the hybrid methods based on evolutionary algorithms will be explained, the advantage of off-policy learning will become substantial. These methods can still learn from past experiences acquired by other agents (or individuals). Off-policy methods are thus the optimum candidates to be integrated into the hybrid frameworks described in Chapter 3.4. Therefore, on-policy exploration is no longer discussed.

### 3.2.2. Learning With Function Approximators

The algorithms for exact value iteration require the storage of distinct return estimates for every state-action pair in the Q-table. Similarly, the policy collects the actions followed in a list-like structure. Furthermore, when the problem has some continuous states or actions (such is the case for aircraft control), the required discretisation introduces additional errors. For the classic RL solutions, the resulting number of state-action pairs increases exponentially with the dimensionality of the state-action space, requiring more computational resources. Hindering the capacity of RL to scale up, this phenomenon is known as the *curse of dimensionality* [5, 54].

To circumvent this problem, the policy and value can be approximated[57]. Reinforcement learning has seen throughout time many possibilities to perform function approximation such as radial basis functions, fuzzy logic systems, least-squares linear regressors and neural networks [57]. For their generalisation power in high-dimensional control tasks, neural networks have become the modern state-of-the-art for function approximators in reinforcement learning [11, 57, 10].

#### Artificial Neural Networks

Artificial neural networks (ANNs), also called feedforward neural networks, or fully-connected neural networks, have one goal: to approximate the function $F$ that maps the recorded input-output pairs. For example, for a classifier, $y = F(x)$ maps an input $x$ to a category or class $y$. A feedforward network defines a parametrised mapping $\hat{y} = f_\theta(x)$ and learns the value of the parameters $\theta$ that results in the best function approximation, or smallest deviation between a set of samples from the $y$ and $\hat{y}$ distributions [10]. These models are called feedforward because information flows through the function being evaluated from $x$, through the intermediate computations used to define $f_\theta$ (called *hidden layers*), and finally to the output $y$.

In the context of reinforcement learning, ANN is a valid and popular surrogate for the Q-function and/or the policy function [58]. For brevity, in this section, $x$ and $y$ will generically denote the NN's input and output vectors or tensors.

#### Field terminology

The term network comes from their mathematical representation. Typically, they are represented by composing or chaining together many different functions. By adapting the example from [10], one can take three functions $f_1()$,$f_2()$, and $f_3()$ connected in a chain, to build the mapping $f(x) = f_3(f_2(f_1(x)))$. In this example, $f_1$ is called the first layer or the *input layer* of the network, $f_2$ is called the second layer or first *hidden layer*, and $f_3$ is the third or *output layer*. The total length of the chain gives the depth of the model.

Historically, the name 'deep learning' (DL) from deep reinforcement learning arose from this terminology but, for the current work, they remain as a formal classification. Also, each hidden layer of the network is typically vector-valued and the maximum dimensionality of these hidden layers determines the width of the model. Both the number of layers and their width is tuned as hyperparameters. Lastly, it is sometimes relevant to consider the total number of neurons which is referred to as the *model footprint*.

The term 'neural' is loosely inspired by the neuroscientific view of the brain, used in an artificial environment. Each element of the vector may be interpreted as playing a role analogous to an artificial neuron. The similarity, although present, is limited and the neural networks should not be seen as models of the biological brain.

**Computational hidden unit**
Going inside the model itself, neural networks are at their core a generalisation of a linear regression created by stacking two or more such affine transformations with some kind of non-linear function, called *activation*, in between [59]. The combination of the linear transformation and the non-linear activation forms the computational unit (neuron); the building block of the network. Going back to the chain of functions from above, any neuron $f_i()$ is described by:

$$f_i\left(\boldsymbol{x}\right) := g_i\left(W_i^\top \boldsymbol{x} + \boldsymbol{b_i}\right) \tag{3.13}$$

The weights $W_i$ and bias terms $\boldsymbol{b_i}$ are the neuron's parameters to be optimised during training. For future reference, the set of learnable parameters of the entire network is denoted with $\theta$. As for the non-linearity $g_i()$, functions such as the sigmoid, hyperbolic tangent, rectified linear unit (ReLU) or exponential linear unit (ELU) are all common choices supported in the mainstream libraries. Both ReLUs and ELUs alleviate the vanishing gradient problem via the identity for positive values thus benefiting from numerical stability during training [10]. Furthermore, [60] showed that ELUs enhance the training speed as they can map to negative values. Throughout this research, the ELU activation was implemented, should the authors of a particular algorithm had not required otherwise.

$$\text{ReLU}(z) = max\{0, z\} \tag{3.14}$$

$$\text{ELU}(z) = \begin{cases} z, & z > 0 \\ \alpha \cdot \left(e^z - 1\right), & z \leq 0 \end{cases} \tag{3.15}$$

**Loss function**
Borrowing from optimal control, the fitted neural network is considered to be the one with the minimum cost value over the available data set. Specialised loss functions enable training networks in a*supervised* manner. The total cost function used to train a neural network is the sum of the losses over all the data samples [10].

In deep reinforcement learning, the data sets are built from past experiences collected in a memory buffer or sampled from the behavioural policy. Moreover, since RL learns based on temporal difference updates, the approximators cannot be trained in the standard supervised manner and their loss values have to be estimated differently. Section 3.2.3 will detail both how DRL accesses past interactions with the environment and how the loss values are calculated.

The task of approximating the value and action functions of an agent represents a regression problem. Therefore, training the function approximator corresponds to minimising the Mean Squared Error (MSE) from Equation 3.16 with respect to the model parameters.

$$\mathcal{L}\left(\mathsf{x}^{(\mathsf{i})}, \mathsf{y}^{(\mathsf{i})}, \theta\right) = \frac{1}{|B|}\sum_{\boldsymbol{x}\in B} l(\mathbf{x}) = \frac{1}{|B|}\sum_{\boldsymbol{x}\in B}\left(F(\boldsymbol{x}_t) - f_\theta(\boldsymbol{x}_t)\right)^2 \tag{3.16}$$

With $|B|$ denoting the number of samples (input-output pairs) contained in the data *batch* $B$ over which the loss function is computed.

Usually, the cost function also incorporates a regularisation term that penalises the complexity of the network. This is employed to improve the model generalisability by preventing it from *overfiting* the training samples. For reinforcement learning applications, the network's width and depth are typically small, in the order of $10^2$ neurons spanning at most $10$ layers. Comparing this to modern deep models that train billions of learnable parameters in hundreds of layers, it seems that RL does not (yet) fully benefit from regularisation terms. Nevertheless, as the field evolves, some DRL algorithms use increasingly complex neural networks and thus reinforcement learning regularisation remains an open research question [20].

**Backpropagation and parameter update**
The deep learning frameworks are commonly referred to as *gradient-based* methods since they require the gradient, or partial derivative, of the cost function with respect to each learnable parameter. One way to achieve this efficiently is the backpropagation algorithm. Basically, starting from the output layer, the partial derivatives of the cost functions are calculated and rolled back one layer at a time. This results in a fast and memory-efficient way to train deep networks which are fully supported in deep learning modules such PyTorch[1] or TensorFlow[2].

In this framework, the intrinsic core assumption states that the loss function shall be differentiable. This is an essential condition for estimating gradients and back-propagating them during training. Nevertheless, reinforcement learning practitioners have realised that the assumption is usually constraining when the aim is to predict the future return of a state-transition pair. When the optimisation problem becomes too complex (e.g., highly non-convex, non-smooth, non-differential, discontinuous, and NP-hard) the resulting gradient signals are sparse and less informative thus damping the effectiveness of gradient-based learning [61, 35]. Chapter 3.3 will detail another method to train the network's parameters that can potentially circumvent these issues.

First, the parameters are initialised randomly with small values that prevent saturation of the non-linear activation functions. Algorithms such as Xavier initialisation use semi-empirical formulae to take care of this step [62]. Then, during training, the gradient values are used to optimise the parameters according to Equation 3.17. Most modern neural network models use adjusted implementations of the stochastic gradient descent optimisation algorithm (SGD), such as the Adam optimiser [63, 29].

$$\theta_{\mathsf{k}+1} = \theta_{\mathsf{k}} - \alpha \cdot \frac{1}{|B|} \sum_{\mathsf{i}=1}^{|B|} \nabla_{\theta_{\mathsf{k}}} \mathcal{L}\left(\mathsf{x}^{(\mathsf{i})}, \mathsf{y}^{(\mathsf{i})}, \theta\right) \tag{3.17}$$

The network weights $\theta$ are updated iteratively with $k$ denoting the current iteration step (or epoch) and $\alpha$ the learning or optimisation rate. The learning rate hyperparameter directly influences the convergence behaviour: a small $\alpha$ results in slow learning whereas a large value can cause divergent behaviour. Thus, tuning it properly is a mandatory step during the algorithm design.

During training, the network parameters are moved towards a local minimum of the loss function. Therefore, within deep reinforcement learning, gradient descent is considered efficient in exploiting local information. Despite the advantage, convergence to local minima results more often than not in sub-optimal performance in the global landscape. By comparing gradient-based learning to a random search in the context of game playing and control tasks, [64] suggests that local optima, saddle points, and noisy gradient estimates are all impeding learning.

**Network architectures**
Apart from the above-described feed-forward neural network, many other architectures prove useful in different learning tasks. Convolutional neural networks have proven useful for processing data with strong spatial correlations, such as visual data [65], whereas recurrent neural networks and attention mechanism (usually incorporated in transformer models) exploit the sequential patterns in text, speech or dynamical systems [66, 67].

The benefits of these specialised architectures arise in the big-data regime [10]. In RL in general and for flight control in particular, it is difficult to collect millions of relevant samples in the agent's experience buffer. Furthermore, the current task of aircraft control does not require processing visual data or learning long-time dependencies. For these reasons, the state-of-the-art DRL algorithms presented in this work involve only relatively shallow feed-forward neural networks (at most 4 hidden layers). Moreover, the learning shall be performed offline in a simulated environment to maximise the number of samples collected and used. Section 3.2.4 will come back to the aspect of offline training.

**Hardware acceleration**
Training of deep neural networks benefits directly from heavily developed hardware such as Graphics

[1]`https://pytorch.org/docs/stable/autograd.html` [Visited on 01/05/2022

[2]`https://www.tensorflow.org/api_docs/python/tf` [Visited on 01/05/2022]

Processing Units (GPUs) or Tensor Processing Units (TPUs). During the forward and backward passes, the batches of samples are stored as matrices or tensors, depending on the shape of the sampled feature space). These hardware units are specialised in performing fast parallel multiplications of such data structures, hence speeding up the training process [68].

Lastly, its unprecedented success in many complex tasks has helped DL reach the mainstream. While the field is still maturing, it has built a strong, diverse and supportive international community both in academia and industry.

**Hyperparameter search**
Deep reinforcement learning algorithms are known to show brittle convergence properties, highly dependent on the settings used [14, 35, 42]. More specifically, the field's literature agrees that the learning performance of a particular DRL algorithm is strongly sensitive to changes in the specific hyperparameters used. One set that achieves the optimum performance in one environment might converge to a sub-optimal policy in another one. Thus, meticulous tuning of the learning rates or discounting factor, exploration parameters and other method-specific settings becomes a necessary condition for each new application.

Alternatively, it has been observed that a particular method can perform much better after extensive tuning. Automated RL (AutoRL) frameworks, such as SEARL [41], provide ways for hyperparameter optimisation to be performed automatically in an external, meta-learning, loop [69]. In practice, meta-learning provides an increase in the algorithm's sample efficiency at a cost of computational resources.

The preliminary phase of the current thesis aims to select a learning framework that performs well in the task of fault-tolerant continuous control. That is why, during the preliminary phase, the resources allocated for hyperparameter tuning are reduced. Nevertheless, a more extensive search for a well-performing set will be conducted in the later stage for the selected application domain (see Part III). For the preliminary analysis later described in Chapter 4 of this part, a performance margin is considered for the investigated algorithms:

**Performance Margin (preliminary analysis**

Two or more DRL algorithms will be considered similar if and only if their final mean performance is within $5\%$ with the hyperparameters recommended by the designer of the algorithm[a].

[a]For most DRL algorithms websites such as SpinningUP contain a set of recommended parameters. If not available, the set from the original paper will be taken.

Whereas the performance gap aims to relax the performance comparison during the preliminary study. More sophisticated methods of statistical analysis should be involved to make decisive conclusions but this goes beyond the scope of the current work.

To allow others to replicate or reproduce the work, it is considered necessary to not only clearly report the values of the hyperparameters but also document the tuning process [70]. Consequently, the performance claims of authors that do not provide these details are harder or even impossible to verify. This has been taken into account during the current literature study while selecting the relevant methods for further analysis.

### 3.2.3. State-of-the-art DRL Algorithms
Now that the framework is introduced, one can take a look at the most relevant algorithms implemented in the literature. As already mentioned, the task at hand concerns continuous control, so only the methods appropriate for continuous state and action spaces are considered. Figure 3.3 inspired from [71] depicts some of the most significant algorithms belonging to the DRL frameworks. Covering all methods and frameworks that have been developed goes beyond the scope of the current work. An empty branch has been added to Figure 3.3 to showcase that its overview is incomplete.

Notably, AlphaZero and MuZero were among the first mainstream successes of DRL. They managed to beat the best human players in complex strategic games such as GO, Shogy and Starcraft[3].

The current work aims to perform offline learning for fault tolerance. Therefore, the model-based frameworks are less attractive as they are prone to overfitting the dynamics model [18, 31]. The right-hand side of the three from Figure 3.3 is thus left out of the scope of future discussions.

As the goal of the current work is to find the synergy between reinforcement learning and evolutionary processes, the hybrid implementations presented in Chapter 3.4 only consider off-policy control for its superior efficiency in learning from past experiences. Therefore, the methods that use on-policy exploration are not considered further, as the focus is on the off-policy algorithms. Lastly, the Hindsight Experience Replay (HER) is applicable whenever there are multiple goals which can be achieved, e.g. achieving each state of the system may be treated as a separate goal [72]. Any off-policy algorithm can benefit from using HER so its implementation is left as a possible future step to improve the performance of the hybrid frameworks described later.

The following sections will detail the DDPG with the DQN and Policy Gradient components and two of its extensions, the TD3 and SAC algorithms.

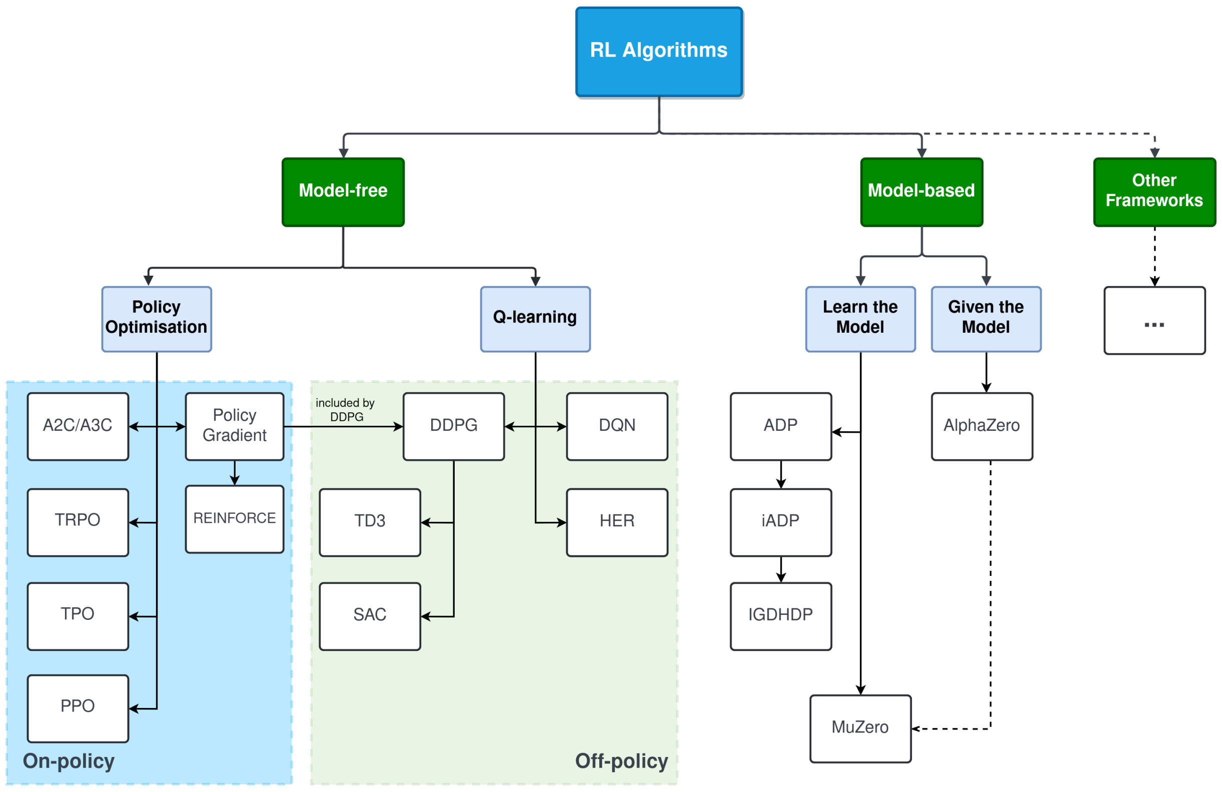


**Figure 3.3:** Modern DRL algorithms visualised in a tree-like structure. The figure is adapted from [71].

### Deep Deterministic Policy Gradient

Policy gradient methods frame the goal of maximising return as the minimisation of the loss function $\mathcal{L}(\theta)$ where $\theta$ symbolises the parametrised agent. A widely used policy gradient method is Deep Deterministic Policy Gradient (DDPG) [12], a model-free RL algorithm developed for working with continuous high dimensional action spaces. It closely resembles the Q-learning approach of Deep Q-Networks (DQN) but it follows a continuous policy. Both will be described below.

*Deep Q-learning*: First, the Q-function $Q(s, a)$ is approximated by a parametrised function $Q_\phi(s, a)$ which is differentiable with respect to its parameters $\phi$.

[3]`https://www.deepmind.com/blog/muzero-mastering-go-chess-shogi-and-atari-without-rules` [Visited on 24/05/2022]

The Q-network $Q_\phi$ trains via gradient descent with backpropagation but without supervision since there are no ground truth labels. Essentially, the MSE loss from Equation 3.16 has to adapt to mirror the Bellman optimality function from Equation 3.11. The new approximation loss is the mean squared Bellman error (MSBE), calculated over the collected buffer of past transitions $\mathcal{D}$:

$$\mathcal{L}(\phi, \mathcal{D}) = \mathbb{E}\left[\left(Q_\phi(s,a) - \underbrace{\left(r + \gamma(1-d)\max_{a'} Q_\phi\left(s', a'\right)\right)}_{\text{target value } y(r,s',d)}\right)^2\right], \quad (s,a,r,s',d) \sim \mathcal{D} \tag{3.18}$$

The term $y(r, s', d)$, represents the output of the target network and it is obtained from a forward pass. Hinting toward the code implementation, $d$ is a Boolean variable that stores the observed value of the environment (or trail) termination criterion. Essentially, $d$ is 1 (true) when $s'$ is the final state, incorporating in the Q-function the knowledge that the agent receives no additional rewards in the future. This trick enables a straightforward implementation of the algorithm.

*Policy gradient learning*: Since the policy is continuous, the greedy action selection from Equation 3.10 cannot be performed. The gradient policy part of DDPG aims to learn a deterministic parametrised policy $\mu_\theta(s)$ which returns the action that maximises $Q_\phi(s, a)$. To optimise the policy network, one can just perform gradient ascent with respect to policy parameters, taking the Q-network estimate over the buffer $\mathcal{D}$ as the objective function.

This idea was first introduced by the REINFORCE algorithm developed by [73] for on-policy iterations but it has since also been applied to off-policy algorithms.

Similarly, the A2C and A3C algorithms also depicted in Figure 3.3, perform gradient ascent to directly maximise the current performance (equal to the value function $V(s_t)$) and the *advantage* of the next action $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$ [74]. In general, the policy optimisation problem then becomes:

$$\max_\theta \mathbb{E}\left[Q_\phi(s, \mu_\theta(s))\right], \quad s \sim \mathcal{D} \tag{3.19}$$

*Actor-critic structure*: The previously described DQN's target network was designed to learn to approximate the Q-value function. This function shows how well the current policy will be performing, essentially being a *critic* of it. Similarly, the learnt parametrised policy that performs the actions is named the *actor*. A new actor step will let the critic update via the temporal difference bootstrapping. Consequently, the updated critic is used to improve the policy. The cycle repeats at every iteration.

The DDPG algorithm concurrently learns a Q-function and a policy, so the standard taxonomy considers it an *actor-critic* method [5]. Now, the basic RL algorithm presented at the beginning of this chapter can be updated for future reference. The actor-critic concept can be visualised in Figure 3.4.

*Replay buffer*: As already mentioned, sufficient exploration is necessary to combat inefficient learning in the presence of sparse gradients or local minima of the loss function. The replay buffer $\mathcal{D}$ of the reinforcement learning agent stores its past experiences in the form of $(s, a, r, s')$ tuples obtained from the policies it followed. This ensures that the off-policy exploration can benefit from the previously learnt behaviours, increasing the algorithm sample efficiency. The replay buffer is directly used to train the actor and critic networks.

*Target network*: The actor and critic network parameters cannot be trained in the standard supervised way. Looking at Equations 3.18 and 3.19, the objective value depends on the same parameters (i.e., $\theta$ and $\phi$) that one tries to train. This chicken-egg problem becomes unstable. The solution proposed is to take a set of parameters which comes close to $\theta$ and $\phi$, but with a time delay. This delayed network, called the *target network*, will thus have to lag the main one [12]. The parameters of the target network are denoted $\phi_{\text{targ}}$ and $\theta_{\text{targ}}$ respectively.

In DDPG-based algorithms, the target network is a version of the main network, updated at the same time with it by low-passing its values with a Polyak average, firstly developed by [75] to improve the convergence of iterative methods. This type of soft update introduces a delay in the updated parameters [12]. In Equation 3.20, the soft-update rule is shown for a target function parameterised by $\theta$:

$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta \tag{3.20}$$

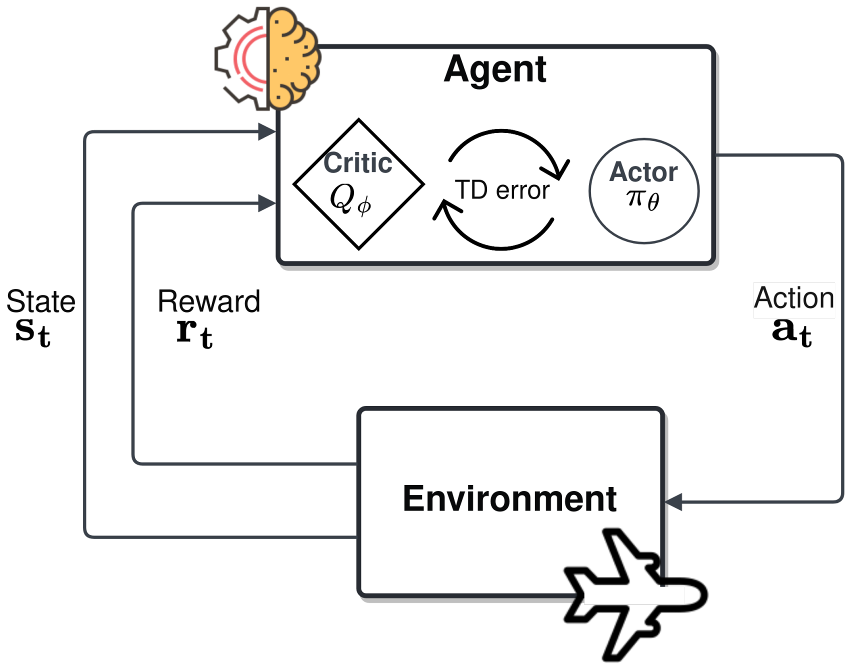


**Figure 3.4:** The agent–environment interaction with a parametrised actor-critic structure.

Where $\rho \in (0,1)$ is a hyperparameter of DDPG. Now, putting it all together, the final version of MSBE loss for DDPG is shown by Equation 3.21:

$$\mathcal{L}(\phi, \mathcal{D}) = \mathbb{E}\left[\left(Q_\phi(s,a) - \left(r + \gamma \cdot (1-d) \cdot Q_{\phi_{\text{targ}}}\left(s', \mu_{\theta_{\text{targ}}}\left(s'\right)\right)\right)\right)^2\right] \quad \text{with} \quad (s,a,r,s',d) \sim \mathcal{D} \tag{3.21}$$

To better understand the exact steps followed by the algorithm, the pseudo-code from Algorithm 1 summarises the implementation of its tricks.

Lastly, DDPG is an off-policy control algorithm as it trains a deterministic policy in an off-policy way. Because the policy is deterministic, should the agent exploit it, it would not try a wide enough variety of actions to even find possibly useful learning signals. To enhance exploration, Gaussian noise was added to the selected actions at training time [12]. As shown by line 3 from Algorithm 1, the resulting actions are then clipped within pre-define action bounds to ensure stability during the interaction with the environment.

DDPG is a common solution as it leverages some of the above-mentioned tricks to improve the sample efficiency of learning but is considered notoriously challenging to use due to its extreme brittleness and hyperparameter sensitivity. The following two algorithms add new tricks to combat these undesirable characteristics.

**Twin-delayed DDPG**

A common failure mode for DDPG is that, after a certain number of training episodes, the learned Q-function begins to dramatically overestimate Q-values, which then leads to the collapse of the exploited policy [13]. Theoretical studies performed by [76] analytically proved that Q-learning suffers from an *overestimation bias*. They argue that the greedy maximisation of a noisy Q-function estimate induces a consistent overestimation even if the approximation noise is zero-mean.

Such inaccuracy will continue to increase due to the recursive nature of temporal difference learning in which an estimate of the value function for one state is updated using the estimate of a subsequent state. This means that using an imprecise estimate within each update will lead to an accumulation of errors. Over multiple updates, the accumulated error can cause arbitrarily 'bad' states to be estimated with a high Q-value, resulting in sub-optimal policy updates and divergent behaviour [13].

The Twin Delayed DDPG (TD3), developed by [13], directly addresses the issue of Q-value overestimation by clipping the outputs of the critic Q-network, smoothing the target policy and further delaying the updates of the target actor-network. The algorithm is highlighted by the pseudo-code from Figure 2.

**Algorithm 1 Algorithm: Deep Deterministic Policy Gradient (DDPG)**

```
    Input: random policy network parameters $\theta$, Q-function parameters $\phi$, empty buffer $\mathcal{D}$
 1: Initialise the target critic and policy networks by the hard update: $\theta_{targ} \leftarrow \theta$ and $\phi_{targ} \leftarrow \phi$
 2: while not converged do
 3:     Observe current state $s$ select clipped action $a = clip(\mu_\theta(s) + \epsilon, a_{low}, a_{high})$ with $\epsilon \sim \mathcal{N}(0, \sigma)$
 4:     Execute action $a$ in environment
 5:     Observe new state $s'$, reward $r$ and bool $d$ indicating if the new state is terminal
 6:     Store experience as transition tuple $(s, a, r, s', d)$ in the replay buffer $\mathcal{D}$
 7:     if $d$ is True then:
 8:         Reset environment
 9:     end if
10:     for $iter$ in $range(max_{iter})$ do
11:         Sample random transitions batch $B = ((s, a, r, s', d))$ from $\mathcal{D}$
12:         Use batch to compute target value:
                $$y(r, s', d) \leftarrow r + \gamma(1-d) Q_{\phi_{\text{targ}}}\left(s', \mu_{\theta_{\text{targ}}}(s')\right)$$
13:         Update critic by one step of batch gradient descent (shown by Equation 3.17) of the MSBE
            loss from Equation 3.16:
                $$\phi \leftarrow \phi - \alpha_{critic} \cdot \nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$
14:         Update policy by one step of gradient ascent:
                $$\theta \leftarrow \theta + \alpha_{actor} \cdot \nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$
15:         Soft-update the target networks using Equation 3.20:
                $$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1-\rho)\phi$$
                $$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1-\rho)\theta$$
16:     end for
17: end while
```

*(Clipped) Dual Q-networks*: Firstly, TD3 learns two Q-functions instead of one, thus the 'twin' name. Then, it uses the smaller of the two Q-values to form the targets in the MSBE loss as shown by Equation 3.22 By involving the less optimistic value function as an optimisation target, the algorithm is less prone to overestimation bias which was empirically shown to improve learning stability [13].

$$y\left(r, s', d\right)=r+\gamma(1-d) \min_{i=1,2} Q_{\phi_{i, \text{targ}}}\left(s', a'\left(s'\right)\right), \tag{3.22}$$

Both Q-networks are trained according to the MSBE loss from Equation 3.21.

*Target policy smoothing*: Secondly, the actions used to train $Q_{\phi_{\text{targ}}}$ are still based on the target policy, $\mu_{\theta_{\text{targ}}}$, but with clipped Gaussian noise added on each dimension of the action. After adding the noise, the target action is clipped again to lie in the valid or safe action range, defined by the interval $[a_{Low}, a_{High})]$. The sampled action thus becomes:

$$a'(s')=\text{clip}\left(\mu_{\theta_{\text{targ}}}(s')+\text{clip}(\epsilon,-c,c), a_{Low}, a_{High}\right), \quad \epsilon \sim \mathcal{N}(0,\sigma) \tag{3.23}$$

In DDPG, if the Q-function approximator estimates an incorrect sharp peak for some actions, the policy will incorrectly exploit that peak. In contrast to just clipping the selected action $a$ (as done by line 4 of DDPG), target policy smoothing can be seen as a form of regularisation.

*Delayed policy updates*: Lastly, empirical experiments suggest that learning failure can occur due to the interplay between the actor and critic updates. Overestimation becomes a problem when the policy is noisy or far from the optimum, and the policy will become poor if the critic's estimate itself is inaccurate. Thus, as the target networks help reduce the error over multiple Polyak updates, and policy updates on high-error states cause divergent behaviour, then the policy network should be updated less frequently than the critic network [13]. The pseudo-code from Figure 2 updates the policy in-between lines 15 and 18.

The delayed rate of updates, $policy\_delay$, becomes a hyperparameter of TD3, always set to be higher or equal to 2.

**Soft Actor-Critic**
The last state-of-the-art algorithm investigated is the Soft Actor-Critic (SAC) [14]. From Figure 3.3, SAC is an off-policy algorithm based on DDPG.

SAC was developed roughly at the same time as TD3 and incorporates some of its tricks. More specifically, SAC, DDPG and TD3 share the following:

1. All three learn Q-functions through MSBE minimisation, by regressing to a single shared target.
2. All update the target Q-networks by Polyak averaging from Equation 3.20 of the Q-network parameters over the course of training.
3. Both SAC and TD3 make use of the clipped dual target Q-network trick to combat the overestimation bias of DDPG.

The most important extension from DDPG and TD3, which learn deterministic policies, is that SAC optimises a stochastic one. Therefore, target smoothing is achieved by different means. While TD3 accomplishes smoothing by adding clipped random Gaussian noise to the next target actions, the noise from that stochastic actor is sufficient to obtain a similar smoothing effect [14].

*Entropy regularisation*: A central feature of SAC is *entropy regularisation*. The policy is trained to maximise a trade-off between the expected return and its entropy, the measure of policy randomness given by Equation 3.24. One might see a direct link to the exploration-exploitation trade-off: increasing entropy results in more exploration, which can improve performance in the long run by evading local optima.

$$\mathcal{H}(P)=\underset{x\sim p}{\mathbb{E}}[-\log \mathcal{P}(x)] \tag{3.24}$$

With $x$ denoting a random variable with probability distribution function P. In entropy-regularised reinforcement learning, the agent essentially receives at one particular time step a *bonus reward* proportional to the entropy of the policy at that time step. Looking back at the recursive Bellman update of $Q(s,a)$ from

**Algorithm 2 Algorithm: Twin Delayed Deep Deterministic Policy Gradient (TD3)** from [13] based on [71]

```
    Input: random policy network parameters $\theta$, critics parameters $\phi_1$, $\phi_2$, empty buffer $\mathcal{D}$
 1: Initialise the target critics and policy network by hard update: $\theta_{targ} \leftarrow \theta$ and $\phi_{targ_{1,2}} \leftarrow \phi_{1,2}$
 2: while not converged do
 3:     Observe current state $s$ select clipped action $a = clip(\mu_\theta(s) + \epsilon, a_{low}, a_{high})$ with $\epsilon \sim \mathcal{N}(0, \sigma)$
 4:     Execute action $a$ in environment
 5:     Observe new state $s'$, reward $r$ and bool $d$ indicating if the new state is terminal
 6:     Store experience as transition tuple $(s, a, r, s', d)$ in the replay buffer $\mathcal{D}$
 7:     if $d$ is True then:
 8:         Reset environment
 9:     end if
10:     for $iter$ in $range(max_{iter})$ do
11:         Sample random transitions batch $B = ((s, a, r, s', d))$ from $\mathcal{D}$
12:         Compute target action:
                $$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$
13:         Compute target value over the batch using the most conservative critic:
                $$y(r, s', d) \leftarrow r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{targ,i}}\left(s', \mu_{\theta_{\text{targ}}}(s')\right)$$
14:         Update each critic by one step of batch gradient descent (shown by Equation 3.17) of the MSBE loss from Equation 3.16:
                $$\phi_i \leftarrow \phi_i - \alpha_{critic} \cdot \nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2 \quad \text{for} \quad i = 1, 2$$
15:         Update policy by one step of gradient ascent:
                $$\theta \leftarrow \theta + \alpha_{actor} \cdot \nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$
16:         if $iter$ % $policy\_delay == 0$ then:
17:             Soft-update the target networks using Equation 3.20:
                    $$\begin{aligned} \phi_{targ_{1,2}} &\leftarrow \rho\phi_{targ_{1,2}} + (1 - \rho)\phi_{1,2} \\ \theta_{targ} &\leftarrow \rho\theta_{targ} + (1 - \rho)\theta \end{aligned}$$
18:         end if
19:     end for
20: end while
```

Equation 3.12, the new entropy term is directly added to the expected return according to Equation 3.25:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}\left[R\left(s,a,s'\right)+\gamma\left(Q^{\pi}\left(s',a'\right)+\lambda H\left(\pi\left(\cdot \mid s'\right)\right)\right)\right] \quad \text{with} \quad s' \sim P, a', \sim \pi \\ &= \mathbb{E}\left[R\left(s,a,s'\right)+\gamma\left(Q^{\pi}\left(s',a'\right)-\lambda \log \pi\left(a' \mid s'\right)\right)\right] \end{aligned} \tag{3.25}$$

Here, the next actions are sampled from the current policy, and not from the replay buffer as done by DDPG and TD3. The hyperparameter $\lambda$ is known as the temperature factor that dictates the relative importance of the entropy regularisation. Similarly to the discounted return, $\lambda$ is always positive but lower than 1 to ensure convergence.

Now, to train the regularised network, the MSBE loss function from Equation 3.18 is again assembled. The target value $y(r,s',d)$ is estimated by Equation 3.26 using the minimum of the two clipped target networks but with the entropy term included:

$$y(r,s',d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{targ,i}}(s',\tilde{a}') - \lambda \log \pi_\theta(\tilde{a}'|s')\right), \quad \text{with} \quad \tilde{a}' \sim \pi_\theta(\cdot|s') \tag{3.26}$$

*Stochastic policy learning*: The policy should, at each state, act to maximise the expected future return plus the expected future entropy. Still, the optimisation looks similar to the problem defined by Equation 3.19. Since for gradient-based learning the policy must be differentiable, the authors use an already known technique in the form of the *reparameterization trick*. The action sample is drawn from $\pi_\theta(\cdot|s)$ by computing a deterministic function of the state, policy parameters, and independent noise [18, 14]. Essentially, the original SAC implementation uses a Gaussian policy, taking an action according to Equation 3.27:

$$a(s,\epsilon) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s) \odot \epsilon\right), \qquad \epsilon \sim \mathcal{N}(0,I) \tag{3.27}$$

The tanh is a squashing function that bounds the sampled actions to a finite interval and the operator $\odot$ represent an elementwise product of two vectors.

Finally, all the above steps are summarised by the pseudo-code from Figure 3.

## 3.2.4. DRL in Flight-control Applications

Among the first successful applications of RL to robotics, it was the application of Approximate Dynamic Programming (ADP) used to control a small RC helicopter. The agent learned how to perform complex manoeuvres such as aerobatic manoeuvres and recoveries [9]. Their experiments showed, for the first time, the potential of RL to perform a complex task in robotics and flight control.

**Online model-based learning**

The ADP performs dynamic programming but incorporates function approximators that are learnt online. This method requires a model of the agent dynamics to be known beforehand. Most often in flight control or robotics, such models are not readily available. In [9], the authors implement a form of *apprenticeship learning* to fit a physics-inspired dynamics model on data recorded during human-piloted flights.

Alternatively, methods such as Incremental ADP (iADP) alleviate the need for a globally accurate model [7]. A highly non-linear system is approximated by a local, time-varying, incremental model, resulting in a low computational complexity at the expense of generalisability.

Belonging to the class of actor-critic methods, the Incremental Global Dual-Heuristic Dynamic Programming (iGDHDP) identifies the dynamic model online and uses two neural networks to approximate the cost-to-go (i.e., Q-function) and the policy [6]. The method improves tracking precision in the vicinity of the current state and remains robust to different initial states as Figure 3.5 from [6] shows. The task concern the longitudinal control of a fixed-wing aircraft which, despite being a small state-action space, it triggers highly non-linear plant dynamics.

For the current work, training is to be performed offline. Nevertheless, certain ideas from online methods can be used in a more relaxed form. In Chapter 4, the extension of online adaptation will be introduced.

**Algorithm 3 Algorithm: Soft Actor-Critic (SAC)** from [14] based on [71]

```
    Input: random policy network parameters $\theta$, critics parameters $\phi_1$, $\phi_2$, empty buffer $\mathcal{D}$
 1: Initialise the target critics and policy network by hard update: $\theta_{targ} \leftarrow \theta$ and $\phi_{targ_{1,2}} \leftarrow \phi_{1,2}$
 2: while not converged do
 3:     Observe current state $s$ and sample action $a \sim \pi_\theta(\cdot|s)$
 4:     Execute action $a$ in environment
 5:     Observe new state $s'$, reward $r$ and bool $d$ indicating if the new state is terminal
 6:     Store experience as transition tuple $(s, a, r, s', d)$ in the replay buffer $\mathcal{D}$
 7:     if $d$ is True then:
 8:         Reset environment
 9:     end if
10:     for $iter$ in $range(max_{iter})$ do
11:         Sample random transitions batch $B = ((s, a, r, s', d))$ from $\mathcal{D}$
12:         Compute target value over the batch using the most conservative critic and entropy regularisation:
            $$y(r, s', d) \leftarrow r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{targ,i}}(s', \tilde{a}') - \lambda \log \pi_\theta(\tilde{a}' \mid s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot \mid s')$$
13:         Update each critic by one step of batch gradient descent (shown by Equation 3.17) of the MSBE loss from Equation 3.16:
            $$\phi_i \leftarrow \phi_i - \alpha_{critic} \cdot \nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s,a) - y(r, s', d)\right)^2 \quad \text{for} \quad i = 1, 2$$
14:         Update policy (reparameterised according to Equation 3.27) by one step gradient ascent of the entropy-regularised value function:
            $$\theta \leftarrow \theta + \alpha_{actor} \cdot \nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \lambda \log \pi_\theta(\tilde{a}_\theta(s) \mid s)\right)$$
15:         if $iter$ % $policy\_delay == 0$ then:
16:             Soft-update the critic target networks using Equation 3.20:
                $$\phi_{targ_{1,2}} \leftarrow \rho \phi_{targ_{1,2}} + (1-\rho)\phi_{1,2}$$
17:         end if
18:     end for
19: end while
```

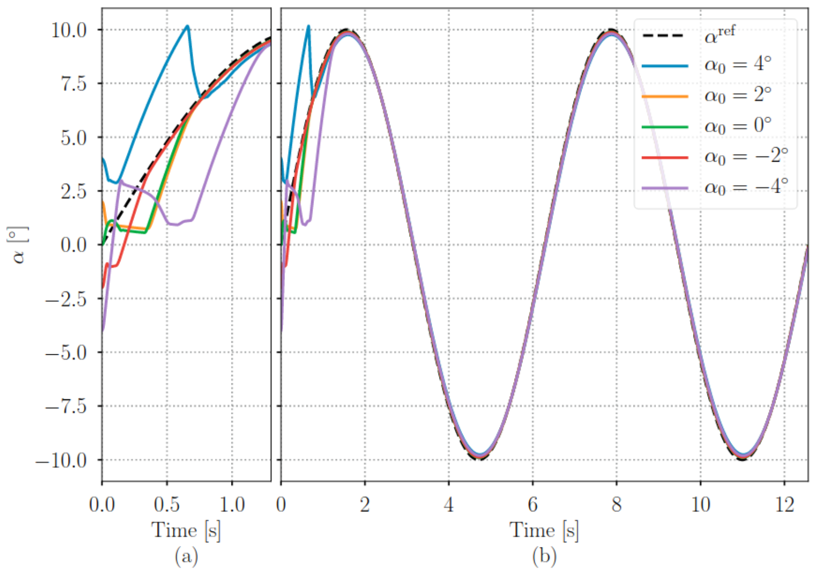


**Figure 3.5:** Angle of attack tracking of an online trained iGDHDP agent [6]

.

**Offline learning**
More recently, model-free offline DRL methods have been applied to flight control tasks. The hope is that such methods could offer a way to tackle the course of dimensionality which inhibits learning for systems with high dimensional state-action spaces.

PPO has been implemented by [19] to control the coupled dynamics of an unmanned small flying-wing aircraft model. The training was performed offline in an episodic fashion and the reward functions were designed to penalise the squared error with respect to reference signals To improve performance, the initial conditions became increasingly more diverse and the reference signals more challenging to follow. Compared to a PID controller, the PPO agent showed similar transient performance in tracking the pitch and roll references whereas benefiting from a smaller steady-state error.

More importantly for the current work, the learnt policy was better able to adapt to wind disturbances and turbulence [19]. As such adverse conditions were not part of the training set, the method showed robustness, being able to generalise to unforeseen circumstances.

Off-policy methods have also become popular. In [16], DDPG is trained offline to control a small fixed-wing UAV. The training set consisted of four different consecutive scenarios, each with 500 episodes, that excite either the longitudinal, lateral or directional modes. Moreover, during each set, the allowable range of one or two actuators was reduced which improved convergence. The simplified action space, despite significantly speeding up the training, limits the precision of the learnt policies in tracking tasks with more than one angle reference.

More advanced algorithms have been applied to both quadcopter and fixed-wing flight control tasks. For example, the go-to-target task is solved in [17] by applying a SAC algorithm with neural inhibitors to train a control policy to optimise the path of a simulated quadcopter. Furthermore, SAC showed promising robustness while controlling a fixed-wing aircraft. In [18], a SAC-based controller was trained offline on the coupled dynamics model of a small jet aircraft. The control architecture consists of two SAC agents in a cascaded framework providing incremental control inputs. The inner-loop agent is trained to provide actuating commands for the aircraft to follow reference signals in pitch, roll and side slip. The outer loop corresponds to an altitude tracker which outputs a pitch signal for the aircraft to follow an altitude reference. The control scheme is shown in Figure 3.6 from [18].

During training, the aircraft is in a nominal condition i.e., no faults have been inflected). After learning from $10^6$ rewarded interactions, the trained policy was able to withstand seven distinct faults and failures such as jammed rudder, reduced aileron effectiveness, structural failure, icing, backward c.g. shift and

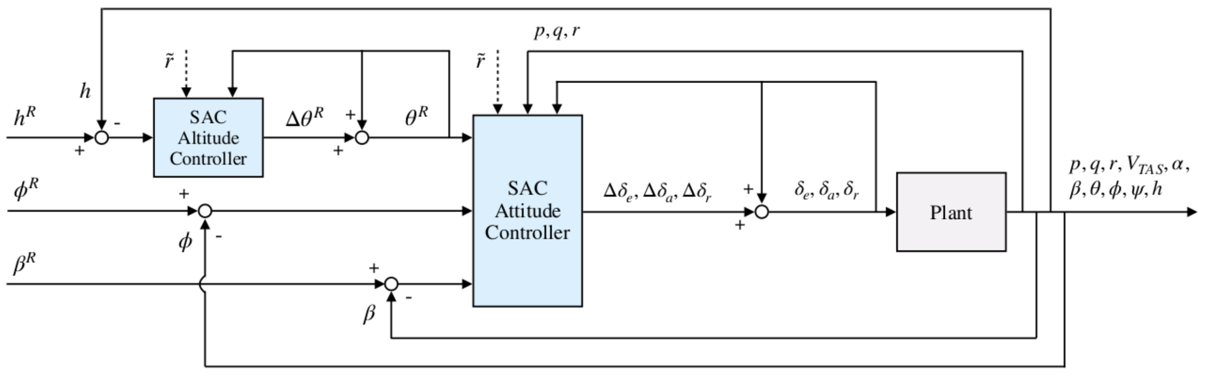


**Figure 3.6:** Altitude and attitude cascaded controllers trained with hierarchical SAC agents. Retrieved from [18].

corrupted sensor measurements [18].Figure 3.7 depicts the response of the agent to the case of a jammed rudder. The failure takes place instantaneously at $t = 10\ s$ but the agents can successfully stabilise the response whereas its tracking error degrades. Then, in the second half of the evaluation episode, another controller engages. This one has been specifically trained on the system in the presence of the aforementioned fault. Successfully commanding the faulty system to follow the references empirically shows that the DRL offline controllers can indeed be extended to an online learning framework.

In general, the above-presented results serve as proof of the potential of state-of-the-art DRL to generalise and remain robust. Despite this, two SAC controllers are needed to control the attitude and altitude separately in a cascaded manner. Also, the incremental control architecture provides smoother action commands which are degrading the convergence rate of offline learning.

Last but not least, other extensions that aim to improve sample efficiency include hierarchical DRL [77] and curriculum learning [78]. While these updates would probably improve tolerance against faults, they do not explicitly treat faulty situations.

Overall, online learning is sample efficient but it performs well only for a limited horizon. During offline learning, the agent can explore more conditions to achieve robust policies. Nevertheless, in the context of fault-tolerant flight control, the main challenge of intelligent controllers learnt offline is how well their policies can generalise to possibly unexpected conditions encountered during flight.

**Computational time**
Where available, the sample complexity of the algorithms described so far in this section is in the order of $10^6$ steps [16, 18]. The reported training times are in the range of a few hours on standard hardware. For example, [19] took took ”about an hour” to train their PPO agent using an Intel i7-9700k CPU and an RTX 2070 GPU.

Despite their performance, the sample complexity of state-of-the-art DRL methods is too high for online learning. Modern DRL algorithms are usually trained offline to benefit from the hardware designed to accelerate deep learning. Aircraft regulations make it harder for them to be equipped with GPUs or TPUs. The small drones usually, although sometimes equipped with newer hardware, cannot provide the required electrical power requested for prolonged periods.

The current work aims to involve offline training on accelerating hardware, so the above-mentioned challenges are not directly constraining. Nevertheless, prospective future tests on real hardware require that the computational constraints shall not be overlooked entirely.

### 3.2.5. Advantages and Limitations
This chapter presented the background behind reinforcement learning. It also detailed how non-linear function approximators can be harnessed to scale up RL to complex, high-dimensional, continuous tasks in continuous control. Three state-of-the-art algorithms were presented, namely DDPG, TD3 and SAC.

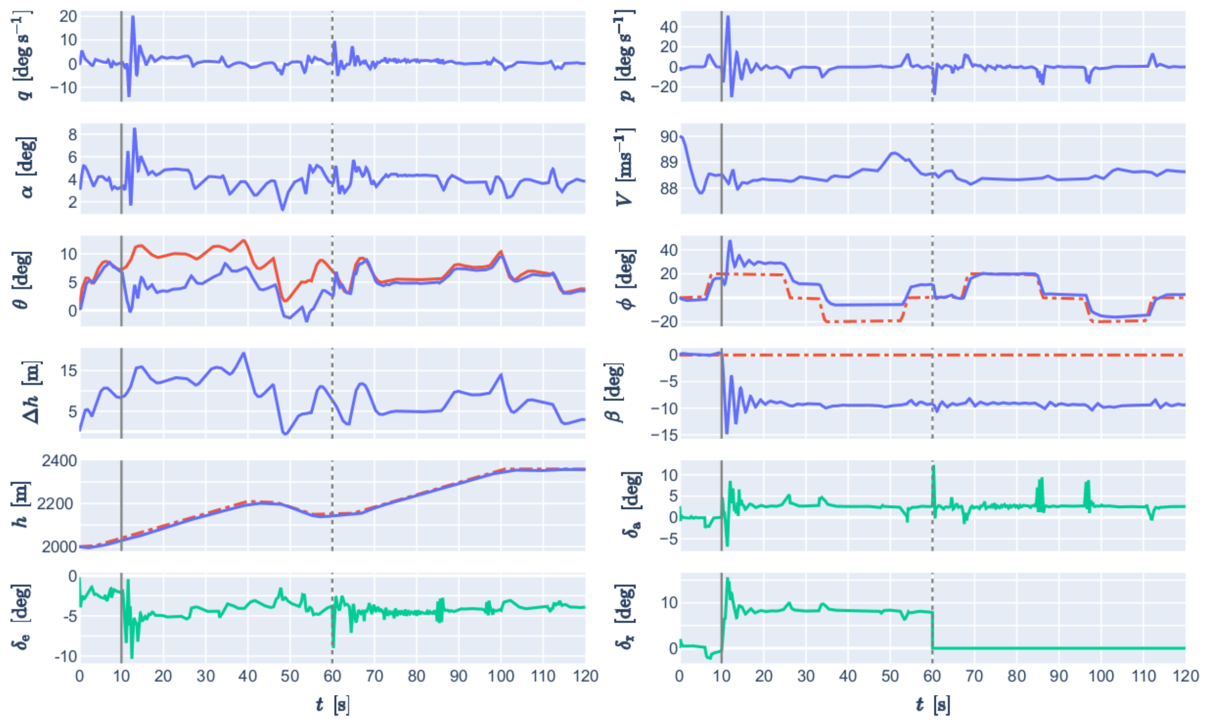


**Figure 3.7:** Altitude tracking response of SAC agents for an aircraft with the rudder stuck at $\delta_r = -15°$ from $t = 10\ s$ (grey solid line). The reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines. The agent trained on the faulty system starts at $t = 60\ s$. Retrieved from [18].

They leverage tricks such as the dual-critic structure, soft and delayed policy updates and entropy regularisation to combat overestimation, enhance exploration and stabilise learning. Then, the focus shifted towards the applications of DRL to flight control. Offline-trained controllers show large potential in terms of generalisation but they require extensive simulation training.

To sum up, let one look back at the identified advantages and disadvantages of deep reinforcement learning. To limit the generality of the discussion, the list of features and drawbacks is compiled whereas targeting the task of fault-tolerant flight control with offline learnt policies.

Answering question **Met-1.a**, the advantages and disadvantages of RL methods are summarised below:

+ The gradient-based optimiser steps for each batch of experiences recorded during the trial, enabling continuous model-free learning over the entire life of the agent.
+ State-of-the-art DRL algorithms such as SAC and TD3 show robustness whereas performing complex tasks in high-dimensional state spaces such as low-level UAV control for reference tracking.
+ Offline training of DRL algorithms benefits directly from heavily developed hardware (GPU, TPU) and a strong, supporting international community both in academia and industry.

The main disadvantages are listed then:

– Intensive exploration is sample-inefficient. Slow learning is due to the presence of sparse gradients or local minima of the loss function.
– Brittle convergence. Optimising a deep neural network critic and/or actor with gradient descent leads to slow convergence of the Bellman optimality equation. The replay buffer is a vital trick but often not sufficient.
– Convergence behaviour is highly sensitive to hyperparameters and to the control architecture.

- Off-policy exploration can lead to unsafe behaviours being learned during training. Such behaviours can become hard to account for in the frame-by-frame learning process.

The next chapter will introduce a new type of optimisation technique that aims to tackle some of these disadvantages.

## 3.3. Neuro-evolutionary Algorithms

Evolution is an alternative to reinforcement learning for iterative or incremental learning methods that aim to solve complex control optimisation problems. The aim of this chapter is to describe the main methods of implementing evolutionary algorithms and reason their advantages and limitations. By doing so, it contributes to the body of research work, answering sub-question **Met-2.c**.

It starts by defining two standard frameworks, in the form of genetic algorithms and evolutionary strategies, described by Section 3.3.1 and Section 3.3.1, respectively. Then, Section 3.3.2 brings forward the concept of quality-diversity, an optimisation framework that encompasses many recent state-of-the-art implementations of evolutionary algorithms. Moving towards relevant practical implementations, Section 3.3.3 reviews a few methods of adapting evolutionary algorithms to cope with faulty systems. Lastly, the chapter concludes in Section 3.3.4 by summarising the main advantages and disadvantages of using evolutionary algorithms

### 3.3.1. Background on Evolutionary Algorithms

As a black-box optimisation technique, an evolutionary algorithm (EA) iteratively improves a *population* of *individuals* based on a *fitness* function. Referring back to the taxonomy presented in the introduction by Table 1.1, they would belong to the policy optimisation family. The fitness function is most often retrieved from previously obtained rewards, so it does not quantify the benefit or value of future actions. It is commonly neither learned nor updated during training so it shall not be seen as an equivalent of the critic network.

Each EA has a few selected component steps. Whereas variations are usual and different frameworks can combine, skip or add certain steps, most evolutionary algorithms are built on the same bare-bone structure. An outline of a standard evolutionary algorithm has been adapted from [79] and it is shown below:

**Evolutionary Algorithm**

1. **Initialisation**: randomly generate the initial population of individuals.
2. **Evolution:** repeat the following steps (generations) until the termination criterion is reached:
   (a) **Evaluate the fitness** of each individual in the population
   (b) **Select** the fittest individuals for reproduction (parents)
   (c) Perform **crossover** and **mutation** operations to generate offsprings of the selected parents
   (d) **Replace** the least-fit individuals of the population with new individuals.

The termination criterion is usually defined as a maximum number of generations or as an expected value of the fitness metric. This can be the maximum fitness within the population, formally referred to as the *elite*, or the average fitness.

It is important to note that the evaluation step calculates fitness by looking at how the individual performed over the entire generation. So, an EA is considered part of the broad class of Monte Carlo learning methods. Whereas this makes EAs invariant to the frequency of actions and reward signals, it does not allow for any in-life updates. Thus, the sparsity of the reward or fitness function is not going to deteriorate their performance. Also, they do not need temporal discounting or any approximations of the value function. Despite this, the update over one generation does not correspond to one epoch of gradient descent. Any subsequent comparison between the training process of an EA and an RL algorithm should be done with caution [80].

Another significant aspect is that none of the EA's steps explicitly need any gradient information. In the literature, the terminology *gradient-free* is applied to such methods whereas, among others, deep reinforcement learning belongs to the *gradient-based* class. The terminology might seem strict but it allows concessions such as the hybrid methods that will be described in Chapter 3.4

In the review study from [81], the authors consider that evolutionary processes are suited for both non-Markov and Markov decision processes. Also, they perform well in tasks that suffer from a lack of observability. Lastly, the policy structures for which gradients are discontinuous or difficult to compute are other drivers of evolution.

Last but not least, individuals can be evaluated in parallel and little to no inter-actor communication is needed. The capacity to parallelise the algorithm enables a higher computational efficiency by scaling to large clusters [35, 29].

As mentioned, the above steps are followed by the vast majority of evolutionary algorithms. The following two sections will present two of the most popular and widely-used implementations of EAs: genetic algorithms and evolutionary strategies.

**Genetic Algorithms**
Genetic Algorithms (GAs) are a class of derivative-free optimisation algorithms that aim to deliver a sufficiently accurate solution within a relatively low computational time. As the first class of evolutionary processes, genetic algorithms are attractive for searching through multidimensional, highly nonlinear non-convex landscapes. In contrast to a gradient-based method, they do not have a tendency to find a local optimum point but search for the global optimal fitness [79].

Genetic algorithms are, by default, domain-independent. Essentially, only the quality of the solution and the simulation time will directly scale with the problem at hand. Despite this, by pouring from an expert-knowledge base, the quality of the optimal solution could be improved. For example, this can be done by involving a specific mutation operator that directs the randomness of the distribution, making it biased towards a specific behaviour [64]. Both Sections 3.3.2 and3.4.1 will further elaborate on this idea.

**Evolving neural-networks**
For the current work, genetic algorithms are interesting as optimisers for a parameterised function approximator. The idea is not, by any means, new, as it dates back to 1993 when it was first introduced by [82].

One of the most influential papers is the Neuro-evolution of Augmenting Topologies (NEAT) [26] which was first proposed in 2002. The original NEAT algorithm aims to solve a reinforcement learning task. It uses a genetic algorithm to optimise the structure and weights of a control policy. The network topology is encoded by a connection graph in which the nodes correspond to the parameters (weights or biases) and the edges are the operations performed with them [27].

NEAT became particularly successful due to its success in addressing the problem of performing crossover networks with variable typologies [24, 25]. The trick is to historically mark the genes with an *innovation number* that stores information on the historical origin of each gene in the population of neural networks. Two genes with the same historical origin must represent the same structure (although possibly with different weights as one can be more evolved), since they are both derived from the same ancestral gene. This information is directly used when crossing over two-parent networks. Also, it prevented premature extinction of augmented structures through a mechanism called *speciation*: divide the population into species such that similar topologies are in the same species. As this is essentially a topology-matching problem, the authors argue that historical markings offer the most efficient solution [26].

To increase innovation, the genome (i.e., the set of genes) is mutated with a certain probability predefined by the algorithm designer. NEAT performs this step in two ways: add a connection or add a node. Novel genes are assigned new increasingly higher innovation numbers [26]. A typical value for the mutation probability is around $0.7 - 0.95$ meaning that each individual has a chance of $70$% to $95$% to be mutated during a generation.

Overall, NEAT managed to prove that GAs are valuable optimisers that can both optimise and train neural networks for complex control tasks. At its time, the usual implementations of neuro-evolution were typically too slow and so NEAT gained significant popularity [24, 25]. Thus, the novel tricks involved in NEAT were very much needed. Despite still benefiting from them, more recent implementations of neuro-evolution using GAs involve more simple encodings, mutation and crossover.

**Recent adaptations**
Algorithms such as the ones used by [64] consider a population of $N$ networks, each encoded by a parameter array $\theta$ still called genotypes). At every generation $i$, each network is evaluated, producing a fitness score $F(\theta_i)$ as a function of its parameters.

The best performing fraction parents are selected as elites which will crossover; their parameters will be combined to form a new network (i.e., the *offspring*). A simple way to implement crossover is the n-point operator: n indexes are picked randomly from the parent genome, and then the values located in between each two selected points are swapped between the parents. Hence, the offspring will share parameter values with both parents. This process can be improved by considering different ways to select the parents [43, 83].

The non-elite population is mutated by applying additive Gaussian noise to the parameter vector:

$$\theta' = \theta + \sigma\epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, I)$$

Also, a fraction of the offspring obtained from crossover can also be mutated. The mutated individuals then replace their predecessors so that the population size remains at $N$. The mutation magnitude $\sigma$ and both the elite and mutated fractions are user-defined hyperparameters.

Lastly, in contrast to NEAT, these new implementations keep the size of networks (width and depth) as hyperparameters are tuned in an outer loop. This is in line with the gradient-based training frameworks which only optimise the weights and biases during the actual training loop.

**Evolutionary strategies**
Evolutionary Strategies (ES) is a sub-category of EAs that apply the evolutionary steps to optimise a vector of parameters that characterise the solution space. Because of that, they were directly involved to optimise both the structure and parameters of function approximators such as NNs [84, 85]. The main differences between ES and GA lie in the representation of the population and the types of evolution operators. Whereas GAs originally used only binary strings, ES has always represented the parameters by a set of real values. For more modern algorithms, this distinction is less applicable as there already have been GAs implementations that also use a real-valued genome to encode the individual [64]. More importantly, ES use mutation operators indirectly in their population-update step, in contrast to GA which explicitly incorporates both crossover and mutation. Empirical studies have proposed that ES are easier to implement and might be faster than GA for optimisation problems with relatively reduced dimensionality.

Within this class, the most widely known member is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [86]. This algorithm represents the population by a full-covariance multivariate Gaussian. Benefiting from the general advantage of random-search algorithms, CMA is robust whereas searching through a rugged fitness landscape, which can comprise discontinuities, (sharp) ridges, or multiple local optima [34]. It is worth noting that, based on these advantages, CMA-ES has so far been extremely successful in solving optimisation problems in spaces with low to medium dimensionality [23, 35].

For the current applications, ES and CMA are used to learn a parametrised control policy. The bare-bones structure of a simplified CMA algorithm is visible in Figure 4 adapted by [35] directly from the original implementation. Essentially, the algorithm follows steps similar to the EAs. First, it initialises the population distribution $p_{\bar{\theta}}$ as an isotropic multivariate Gaussian with mean $\bar{\theta}$ and fixed covariance $\sigma^2 I$. In contrast to the most general versions of CMA, the isotropic distribution assumes an uncorrelated deviation of the parameters.

With this in mind, one can rewrite the fitness function as a stochastic objective which has been blurred by a Gaussian random variable. This surrogate objective is expected to be smoother than the original one [35]. The new objective is given by Equation 3.28:

$$\mathbb{E}_{\theta\sim p_{\bar{\theta}}} F(\theta) = \mathbb{E}_{\epsilon\sim N(0,I)} F(\theta + \sigma\epsilon) \tag{3.28}$$

The stochastic gradient ascent step can be approximated with a covariance-based update of $\theta$:

$$\nabla_\theta \mathbb{E}_{\epsilon\sim N(0,I)} F(\theta + \sigma\epsilon) = \frac{1}{\sigma}\mathbb{E}_{\epsilon\sim N(0,I)}\{F(\theta + \sigma\epsilon)\cdot\epsilon\} \tag{3.29}$$

To improve the approximation, the number of samples $n$ shall increase.

Overall, for each generation, the algorithm executes the following: sample the distribution and perturbed the parameters, evaluate the objective value of the population and update the parameters with a

**Algorithm 4 Algorithm: Simplified Covariance Matrix Adaptation (CMA) [35]**

```
   Input: Learning rate $\alpha$, noise standard deviation $\sigma$, population size $n$,
   initial policy parameters $\theta_0$
1: for $t$ in $range(max\_epochs)$ do
2:     Sample $\epsilon_{i=1,..,n}$ from $\mathcal{N}(0, I)$
3:     Compute fitness return of evaluation:
           $f_i \leftarrow f(\theta + \sigma\epsilon_i) \quad i = 1, .., n$
4:     Update policy via the approximated stochastic gradient ascent from Equation 3.29:
           $\theta_{t+1} \leftarrow \theta_t + \alpha \cdot \frac{1}{n\sigma} \sum_{i=1}^{n} f_i \epsilon_i$
5: end for
```

step proportional to the inverse of the covariance $\sigma$. After sampling from the random distribution, the last step replaces the variation operators used by GAs.

Essentially, CMA uses a finite-difference-like estimation of the gradient. As CMA does not require an analytical formula of the gradient, it will be considered a *gradient-free* optimisation algorithm [87, 35].

### 3.3.2. Quality-Diversity

So far, the optimisation presented considered a population of well-performing solutions but ultimately aimed to select the offspring performing the best in the environment. Unfortunately, that can prove to limit once the rules of the environment change or the agent is forced to behave differently. Due to reasons common with RL frameworks, evolving strategies solely based on their current fitness might prove to be insufficient to achieve the desired performance in the long-run [21, 31].

#### Case for novelty

In novelty search (NS), the idea is to completely ignore the reward or fitness function and only look for increasingly novel solutions [28]. As a simple example, let's consider a four-legged robot that should learn how to walk. While learning the control policy, the first few trials will consist of the robot struggling to move on the ground and failing to stand up. At that point, a novel behaviour would be to stand up. By continuously targeting such novelty encourages, learning emerges in an indirect manner [21]. Conversely, the exploration of the behaviour space is directly and explicitly rewarded.

Efficient exploration is always needed but never sufficient by itself. Also, completely neglecting any fitness or reward metric can result in a time-consuming endeavour [29]. Starting from the idea of novelty search, an algorithm that can explicitly balance the exploration of new behaviours and exploitation of the high-performing ones is considered. This framework is known as quality-diversity search (QD). QD has been formally defined by [30] as an 'optimisation algorithm aiming to produce a large collection of solutions that are both as diverse and high-performing as possible, which covers a particular domain, called the space of behaviours'.

More specifically, algorithms classified within the QD paradigm aim to uncover as many diverse behavioural niches as possible, but where each niche is represented by a candidate of the highest possible quality (i.e., fitness) for that niche [29]. The obtained result maps behavioural space, illuminating it with the best of all the diverse possibilities that exist [87, 88]. Overall, the balanced drive towards increasing diversity with localised searches for quality can be seen as mirroring nature where, traditionally, species face the toughest competition from within their own reign.

#### Behavioural space

In the framework of QD, a particularly key aspect is the definition of behavioural space. As explained by [21], the experimenter has to select some subset of behavioural features that are of interest to form a *behaviour characteristic* (BC) or behaviour descriptor [21]. The span of all BCs forms the behavioural

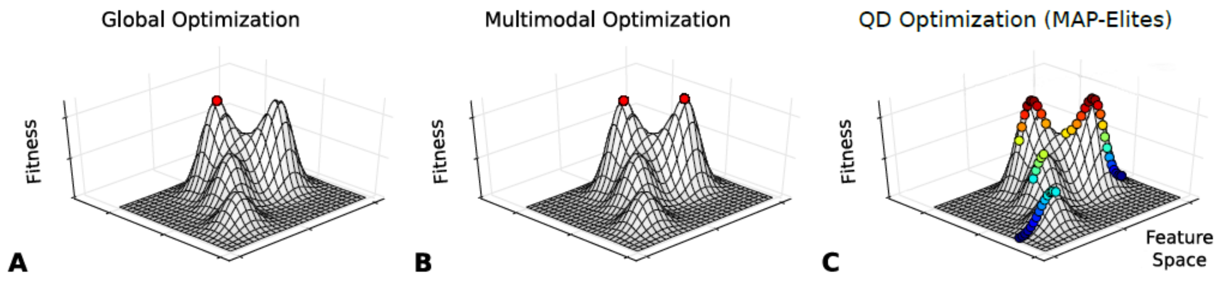


**Figure 3.8:** Visualisation of algorithms performing (A) single-objective global optimisation, (B) multi-objective optimisation aiming at discovering multiple optima and quality-diversity (C) that finds many more solutions (up to three orders of magnitude), each one being the elite of its local neighbourhood defined in some feature space of interest. For the current methods, the feature space corresponds to the behavioural space. Figure retrieved from [90].

space. The selected set of BCs is not unique but a core assumption of QD states that all parts of the behaviour space are considered equally important [21, 87, 29]. In that way, less performant behaviours can still be exploited. Selecting a relevant set of characteristics becomes one of the main goals of the experimenter who has now the opportunity to inject expert-level knowledge into the learning algorithm. As it is directly relevant for any practical implementation, the design of BCs will be further detailed in Chapter 4. Finally, a grid is placed over the behavioural space, dividing it into discrete niches or bins. Then, the task of QD is to maximise a quality score (equivalent to the fitness) within every bin [87].

Formally, QD aims to find the solution to an extended version of the optimisation problem that has been presented so far. Firstly, the QD objective function is designed to return both the fitness value of the parametrised policy $F_\theta$ and its behavioural characteristic $\boldsymbol{b}_\theta$ [89, 29]:

$$f_{QD}(\boldsymbol{\theta}) \rightarrow F_\theta, \boldsymbol{b}_\theta \tag{3.30}$$

Then, one defined the behavioural space by $\mathcal{B}$ and aims to solve the following problem with respect to control policy parameters $\theta$:

$$\begin{aligned} \forall \boldsymbol{b} \in \mathcal{B} \quad \boldsymbol{\theta}^* &= \text{arg} \underset{\boldsymbol{\theta}}{max} F_\theta \\ \text{s.t.} \quad \boldsymbol{b} &= \boldsymbol{b_\theta} \end{aligned} \tag{3.31}$$

The QD optimisation aims at finding each policy with behaviour $\boldsymbol{b} \in \mathcal{B}$ that maximises the fitness value $F_\theta$. The set of policies that solve the optimisation is referred to $\mathcal{M}$.For any, both their fitness and behaviour characteristics are the direct results of following the control policy. In other words, the set of $\boldsymbol{b_\theta}$ describes **how** the solutions solve the problem and fitness value $f_\theta$ measures **how well** the solution performs [89].

For example, considering the control task of landing an aircraft, the fitness could be proportional to consumed fuel (with a negative sign as one should aim to maximise fitness and reduce fuel) whereas the behaviour characteristics would correspond to the vertical speed location along the runway at the moment of touchdown.

Despite prioritising both novelties of the niches of policies and their performance, QD is different from multi-objective optimisation. The relationship and distinctions between these problems are visible in Figure 3.8.

**Multi-dimensional Archive of Phenotypic Elites**

The field of QD shows a relatively large and still increasing popularity but most of the successful methods trace back to the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) algorithm first introduced by [21] as a method to design a set of robust control policies for walking robots. The authors argue that robots, aiming to learn to deal with damage while interacting in a complex environment, have to adapt quickly and efficiently. Before the agent is deployed, it uses a novelty-search technique to create a detailed map (called *archive*) of the space of high-performing behaviours (the *elites*). This map represents the agent's prior knowledge about what policies will result in which behaviour and their relative quality. For this framework, the set $\mathcal{M}$ will include the elites found by the QD optimisation and stored in the archive.

In their approach, once the robot is damaged, it uses this learnt prior archive of knowledge to guide a trial-and-error search algorithm. The damage adaption part will be detailed further in Section 3.3.3. The novelty of this approach stems from the lack of need for diagnosing online or anticipating the failure mode [21]. The agent can access offline-learnt policies and compare their behaviour and quality, both stored in the archive, enabling straightforward adaptation online trials.

Starting from the MAP-Elites algorithm, the quality-diversity framework has seen a recent spree of improvements. The benchmark has been set by the MAP-Elites [21] and, so far, these developments could be split into three categories:

1. Reduce the wall-clock time by parallelisation [91, 87, 92]
2. Improve sample efficiency [31, 93]
3. Combine with other frameworks, such as standard DRL (SAC, TD3), for higher-quality scores [94, 95].

The first category manages to show relevant progress as it usually leverages better computing resources such as clusters of CPU, GPUs or TPUs, all of which have already proved beneficial to speed-up training of deep neural networks [96, 68]. Despite this, it seems that none of the approaches from (2) and (3) provides definitive results. Overall, there will always be a trade-off between the quality of the control policy (i.e., the achieved return) and the diversity of the learnt behaviours (the achieved behaviour characteristics or skill sets).

Model-based frameworks such as [31] show a large benefit by reducing the number of environment interactions by one or even two orders of magnitude. The downside comes from a reduced generalisation power. When tested online on a real system with faults (a spider-like robot with one broken leg), the offline learnt dynamics-aware policies performed poorly, showing a comparable success rate to a simple (vanilla) QD algorithm. As the learnt dynamical model loses its validity once the system changes, its inaccurate predictions deteriorate the controller's performance. A potential solution, also hinted by the authors of [31], would be to incrementally learn the model online, to fit the damaged system. This would increase performance by using more samples from the robot-environment interaction. The aspect of online adaptation will be further discussed in Section 3.3.3.

**Covariance Matrix Adaptation for Map-Elites**
Whereas MAP-Elites show a diverging search behaviour oriented toward multiple reasonably good solutions, algorithms such as the stochastic optimiser CMA-ES (see Section 3.3.1) have the ability to rapidly navigate the low-dimensional behavioural space, aiming for the global optimum or a solution within its neighbourhood [34].

Leveraging the advantages of both, the algorithm developed by [87], takes the selection and branching from CMA-ES to optimise the solutions stored in the MAP-Elites archive. Essentially, the CMA-ME (Covariance Matrix Adaptation for Multi-dimensional Archive of Phenotypic Elites) from [87] directly incorporates the trade-off between exploration and exploitation in the behavioural space. Moreover, the poor scalability of the CMA (mentioned by [35] and [61]) is mitigated since the archive of behaviour always sits in a low-dimensional manifold [21].

CMA-ME is essentially a round-robin scheduler for a population of *emitters*. Each of these is a CMA-ES optimiser that searches through the behavioural space [87]. The pseudo-code of the CMA-ME is shown below:

The routines $generate\_solution()$ and $return\_solution()$ correspond to the structure of the emitter in use. Like CMA-ES, each emitter maintains a sampling mean, a covariance matrix and a parameter set that contain additional CMA-ES-related parameters (e.g., evolution path). However, whereas CMA-ES restarts its search based on the best current solution [34], emitters are differentiated by their rules for restarting and adapting the sampling distribution, as well as for selecting and ranking solutions. The paper proposes three types of emitters to drive the exploration according to a specific intrinsic motivation. The *optimising* emitters use CMA-ES to sample solutions for maximal performance, It is the closest to a standard CMA-ES but it restarts the sample mean and covariance from the location of an elite rather than the fittest solution discovered so far [87]. Next, the *random direction* emitters favour solutions that are as far as possible along a randomly generated direction in the $\mathcal{B}$ space, prioritising behavioural novelty.

**Algorithm: CMA MAP-Elites**

```
Input: An evaluation function $f_{QD}()$ that returns a behaviour characteristic, fitness, and a desired number of solutions $N$
Output: Generate $N$ solutions storing the elites in the archive map $\mathcal{M}$
1: Initialise population of emitters $E$
2: for $sol$ in $range(N)$ do
3:     Select emitter $e$ from $E$ which has generated the least solutions out of all emitters in $E$
4:     $\theta_i \leftarrow generate\_solution\ (e)$
5:     $\mathbf{b}_i, F_i \leftarrow f_{QD}(\theta_i)$
6:     $return\_solution\ (e_i, \theta_i, \mathbf{b}_i, F_i)$
7: end for
```

Lastly, the *improvement* emitters reward additions to the collection of elites $\mathcal{M}$ (i.e., improve the current elites but necessarily add new behaviours).

The algorithm design can harness the potential of each emitter type to specialise the optimisation towards its goal: optimising emitters can accelerate the coverage whereas random direction emitters might escape faster the neighbourhood of a local optimum [89].

Chapter 4 will show the results of training policies using the CMA-ME algorithm to control an agent in a toy-like domain. Moreover, its robustness to system faults will be tested. The tests will only concern offline learning and evaluation. Apart from them, the next section will look into possible ways to use the archive of elite behaviours found by QD to enable online adaptation to damage.

### 3.3.3. Performance for Fault-tolerant Control

The QD algorithms have been proven successful in controlling damaged agents. The original paper of Map-Elites trains the control policies offline, in a simulation environment [21]. Then, the framework uses an Intelligent Trial & Error (ITE) routine, based on the Monte-Carlo tree search algorithm, to sequentially evaluate the control policies on a real but damaged robot. Each evaluation updates the archive of behaviours, subsequently informing the next try. This method achieves an unprecedented performance on real-hardware platforms, successfully bridging the gap between simulation and reality [21].

The work done by [21] showed for the first time the potential of using a repertoire of offline-trained policies to guide an online adaption to unforeseen circumstances and changes. The concept was developed further by frameworks such as [31] which aim to speed up the offline training step by learning a model of the agent-environment interaction or by [36] which perform QD optimisation hierarchically. These subsequent studies agree that the adaptation capabilities are directly linked to the behavioural diversity in the repertoire. This idea was further investigated and expanded upon

Unfortunately, the online adaptation requires manual resets of the agent. Whereas this can be enabled for walking robots, an aircraft cannot be put on hold to reset its controller. Thus, for the current application of flight control, a reset-free algorithm is required for online adaptation. One such is implemented by [33] to solve a navigation task for walking robots by splitting the route into multiple segments and evaluating the policies on each segment.

The work was further expanded by [32] which uses the offline learnt behaviours as priors in a Bayesian learning framework. Then, during the deployment of the real robot, a Gaussian process (GP) model learns to transform the prior behaviours in such a way that the outcomes of policies followed by the real robot match with the newly transformed behaviours. Essentially, it learns how to model the difference between the state transitions recorded during the simulation and the ones observed online.

This framework provides a strong starting point to develop an online adaptation algorithm to select a well-performing policy. Figure 3.9 (retrieved from [32] shows the basic idea behind repertoire-based learning with online adaptation.

### 3.3.4. Advantages and Limitations

Throughout this section, both GA and ES have shown features useful for exploration within the pace of solutions through crossover, mutation and fitness-based selection. As a direct application of an ES

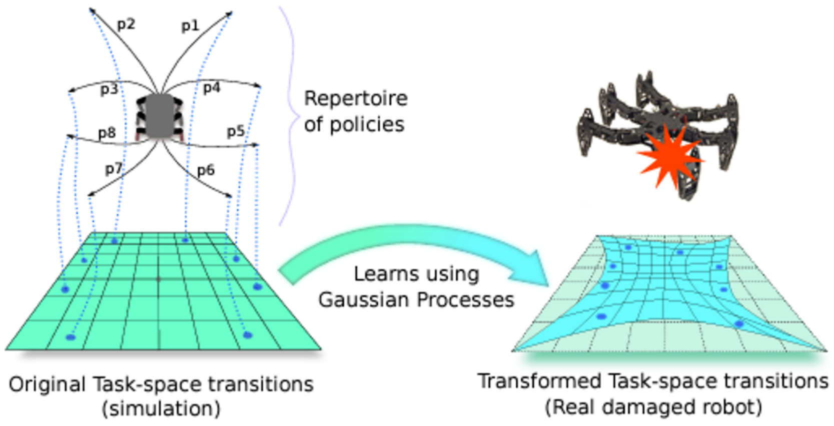

p1
p2
p3
p4
p5
p6
p7
p8
Repertoire of policies
Learns using Gaussian Processes
Original Task-space transitions (simulation)
Transformed Task-space transitions (Real damaged robot)

**Figure 3.9:** High-level visualisation of the repertoire-based learning in robotics. Retrieved from [32].

algorithm, QD benefits from the same advantages. It also provides a more intuitive way to visualise the exploration-exploitation trade-off via the archive of behaviours.

The literature surveyed in this chapter showed that the EA framework is remarkably effective because:

+ Can optimise highly non-convex functions with sparse gradients by escaping local optima or saddle points.
+ As evaluation assesses the performance of the individuals over a generation (i.e., no in-life evaluation), they are invariant to action frequency and delayed rewards and do not require any type of temporal discounting.
+ Will always generate locally optimised repertoires or populations of behaviours mapped by control policies (e.g., robots walking in all directions) that are close to the optimum solution.
+ The agents can be parallelized to run and evolve as parallel workers. It allows for the use of large clusters of CPUs, TPUs, and GPUs, which benefits the wall-clock time required during training.

These advantages make evolutionary algorithms particularly interesting for damage mitigation and recovery in autonomous systems [21, 31, 36]. Furthermore, other successful applications proposed by the literature are procedural content generation, aerodynamic design optimisation and scenario generation in human-machine interaction. The current work will limit itself to the case of fault-tolerant control.

In spite of their convincing advantages, the methods belonging to the EA framework come with several shortcomings:

– They can only update the entire population once the generation, or learning episode, has been completed. Therefore they cannot benefit from the individual's experiences and thus have reduced sample efficiency.
– During the evaluation step, they require a high number of queries of the objective function.
– Poorer scaling to high dimensional spaces in comparison to gradient-based learning.
– The QD framework is specifically sensitive to the choice of behaviour characteristics, the low-dimensional space of behaviours being strictly determined by the learning task at hand.

In conclusion, as a summary of the chapter, the two lists provide an answer to research question **Met-1.b**. One might realise that the advantages and limitations of evolutionary and deep reinforcement learning algorithms do complement each other. It might be possible to combine the two and maybe obtain the best of both worlds. The next chapter will introduce a novel framework that builds upon this idea.

## 3.4. Hybrid Methods

People have long recognised that the exceptional adaptability of the brains of living species results from at least the combined effect of two phenomena. For starters, they are capable of learning from their own life experiences. Second, biology has been assembling an ever-improving set of individuals through evolutionary mechanisms. The successful genomes, in turn, are more capable of further learning, and thus the loop continues. There is hope that the same two factors, combine in a hybrid AI method, will help agents learn more performant and robust behaviours.

This chapter will look into the background, motivation and achievements of the methods belonging to the hybrid class. They are commonly referred to as Evolutionary Reinforcement Learning (ERL). Thus, the following sections will aim to answer questions **Met-2.c** and **Met-3.c**.

First, the historical first method and its results will be summarised in Section 3.4.1. Next, Section 3.4.2 will list the state-of-the-art extensions of the ERL framework. Then, Section 3.4.3 will detail the most promising approach in the form of Proximal Distilled ERL and its results on the metrics available from the surveyed literature.

### 3.4.1. Evolutionary Reinforcement Learning

Even before RL and EA had reached their current popularity, AI researchers realised that the adaptability of living species stems from the effect of the self-development and the learning mechanism of an individual combined with evolutionary processes acting at the population level [37]. This idea sparked the interest of scientists who tried to replicate it in early AI research [38]. The early stepping stones paved the way toward the two formal frameworks described in the previous chapter. Despite these independent developments, research dedicated to the hybrid methods combining both evolutionary algorithms and reinforcement learning has remained active according to [37].

The concept of evolutionary reinforcement learning is mentioned by [97] as a solution to the helicopter hover task in a model-free manner. The authors use a neuro-evolutionary algorithm based on genetic programming formulation to evolve a population of control policies for stable hovering. In contrast to the title of the paper, a hybrid method is not involved.

In [39], NEAT is combined with Q-learning to achieve sample-efficient reinforcement learning whereas [40] uses neural evolution to improve exploration in stochastic domains. Last but not least, EAs have been used to optimise the hyperparameters of DRL algorithms, successfully improving their sample efficiency [41].

Currently, these hybrid methods benefit from unprecedented momentum. It is mainly caused by the same factors sustaining the rapid development of deep neural networks: parallelisation on accelerated hardware and abundance of training data.

#### Recent approach

One of the main algorithms at the origin of the present wave of combining evolution and RL is Evolutionary Reinforcement Learning, proposed by Khadka and Tumer in 2018 [42]. At the same time, the Genetic-gated Networks (G2N) were published by [98] which harvest the benefits of gradient-free optimisation to improve the training performance of deep neural networks via selective drop-out. The authors use these networks as actors and critics in on-policy learning algorithms such as PPO. Their implementation failed to gain traction in comparison to ERL which will be explained in the next section.

Methods within this novel family combine an evolutionary loop with a reinforcement learning loop. In this work, the authors instantiate ERL by combining a standard GA with a DDPG agent. Any off-policy reinforcement learner that belongs to the actor-critic taxonomy could be used but the current section will limit itself to the algorithm used by [42]. Chapter 4 will comment on the future directions and extensions of the presented hybrid frameworks, including the possibility of using other reinforcement learning algorithms.

#### ERL framework

Figure 3.10, inspired directly from the original ERL paper of Khadka and Tumer [42], shows the main steps of the algorithm. To aid the discussion, they can also be followed below:

1. **Initialisation:** population of identically sized NNs is initialised with random weights. In addition to the actor population, one additional network (referred to as $rl_{actor}$) is initialised alongside a critic network.
2. **Loop over generations** until the total number of frames played is reached:
   (a) **Population Evaluation:** the genetic population of actors (i.e., except the $rl_{actor}$) are evaluated by interactions with the environment and the experiences stored in the common memory buffer. The fitness for each individual actor is computed as the cumulative sum of the *undiscounted* reward received during that episode.
   (b) **Selection:** based on their fitness, select a fraction of the population to survive, referred to as the *elites*. The elite fraction is usually set between $0.1$ to $0.3$.
   (c) **Mutation**: the non-elite actors are perturbed by mutating the parameters of the NNs with zero-mean Gaussian noise.
   (d) **Train the RL agent**: follow the steps of the DDPG algorithm to update the critic and $rl_{actor}$ (see Section 3.2.3).
      i. Sample transitions from the common experience replay buffer.
      ii. Compute the output $y(r, s', d)$of the critic network.
      iii. Perform the gradient descent step on the MSBE loss.
      iv. Update the parameters of $rl_{actor}$ by a gradient ascent step.
      v. Polyak update the target networks [75].
   (e) **Synchronisation:** the $rl_{actor}$ is injected into the population by changing the weakest individual.

The GA and RL loops can be directly identified. Whereas the GA enriches the memory buffer, the RL actor is injected into the population at a specific rate.

Each individual within the population is parametrised by a neural network, encoded by a list of real numbers. The order of the list is arbitrary but constant throughout the population. The encoding is similar to the one described by Section 3.3.1.

**Exploratory behaviour**
Exploration is directly achieved by the variation applied to the actors via the mutation operator. During mutation, the weights (genes) of the non-elite individuals are randomly perturbed by additive zero-mean Gaussian noise. The strength of the noise is referred to as *mutation magnitude* and is one of the added hyperparameters. At the same time, the $rl_{actor}$ still explores by adding zero-mean Gaussian noise to its actions.

It is important to also note that, in contrast to more standard implementations of GAs, the first implementation of ERL did not include yet any crossover operator. The authors added the n-point crossover in a future iteration of the method which has also been used during this thesis [42].

**Experiences reuse**
Apart from the mutation operator, one of the main tricks the ERL algorithm introduces is the shared memory buffer. It directly enables the information experienced during the evaluation of the step of evolution to be exploited by the RL agent. In contrast to a standard EA which would extract the fitness metric from these experiences and disregard them immediately, ERL retains them in the buffer and engages the $rl_{actor}$ and critic to learn from them repeatedly using the gradient updates. This mechanism maximises the extraction of information from each individual experience and thus it leads, according to the authors, to improved sample efficiency.

**Synchronisation**
By periodically copying the weights of the $rl_{actor}$ into the genetic population, the information learned during the episode is injected into the population. This is the core mechanism that enables the evolutionary framework to directly leverage the information learned through gradient descent. The process of infusing policy learned by the $rl_{actor}$ into the population also serves to stabilise learning and make it more robust to deception (i.e., local optima). A well-performing policy will survive the evolutionary processes and subsequently influence future generations. However, if the $rl_{actor}$ is less performant, it will simply be discarded after the next evaluation.

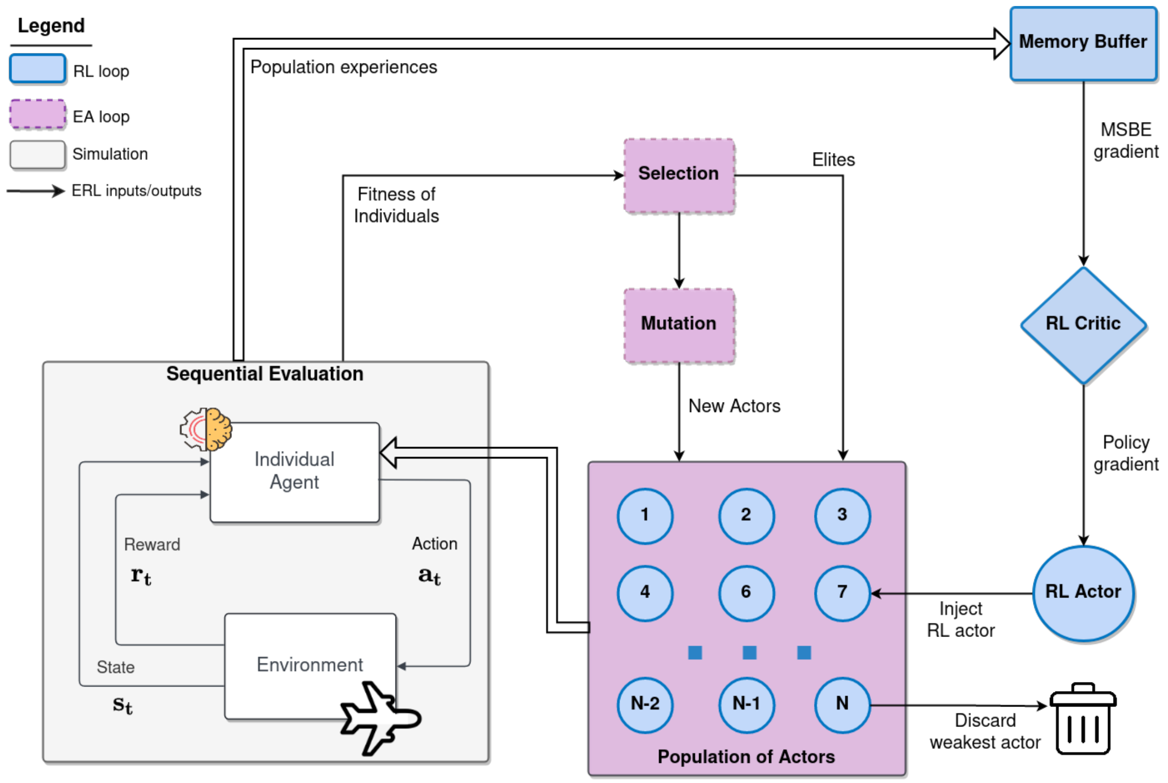


**Figure 3.10:** ERL high-level visualisation highlighting the EA's population-based learning (purple) with DRL's gradient-based optimisation (blue). Adapted from [42] based on [37].

**Results**
The hybrid algorithm was tested on the continuous control environments available in the MuJoCo library, hosted by the OpenAI gym [99]. The six control tasks represent industry-level benchmarks widely adopted by the RL community. Their complexity in terms of action-space dimension and reward structure varies, with Hopper and Swimmer-2d being the easiest ones and Ant-v2 the most challenging. In Figure 3.11, ERL is compared to DDPG, a standard neuroevolutionary algorithm (referenced as EA in the plots) and the state-of-the-art on-policy algorithm PPO [15].

One can see that, ERL can significantly outperform DDPG across all the benchmarks. Similarly, ERL also consistently outperforms EA across all tasks apart from the Swimmer environment, where they achieve similar performances. This environment encourages the agent to hold balance by continuously giving a small constant reward. Since EA selects for episode-wide fitness, this specific reward signal creates a strong local optimum for a policy that simply survives by balancing while standing still. This is the same behaviour EA converges to, hence its relatively high final performance [42].

Overall, the ERL algorithm manages to beat other state-of-the-art frameworks while its return converges to a relatively low variance. This shows that the inherent idea behind is successful. Incorporating gradient-based learning on the in-episode experiences with population-based evolutionary selection and mutation will enhance performance.

## 3.4.2. Extensions of ERL

Despite its clear success, ERL has still limitations that should be addressed. Firstly, it has not fully solved the scalability problem that GAs show. Secondly, whereas the gradient information from the RL agent can significantly speed up the evolutionary search, the population of ERL still evolves using variation operators. The mutation operator, acting on NNs directly encoded as lists of real numbers, can cause destructive learning behaviours, hindering policy convergences [43, 100]. Thirdly, by not using crossover, non-elite individuals might not pass their genes to the next generation.

In the past four years since ERL had first been published, multiple extensions have aimed to address

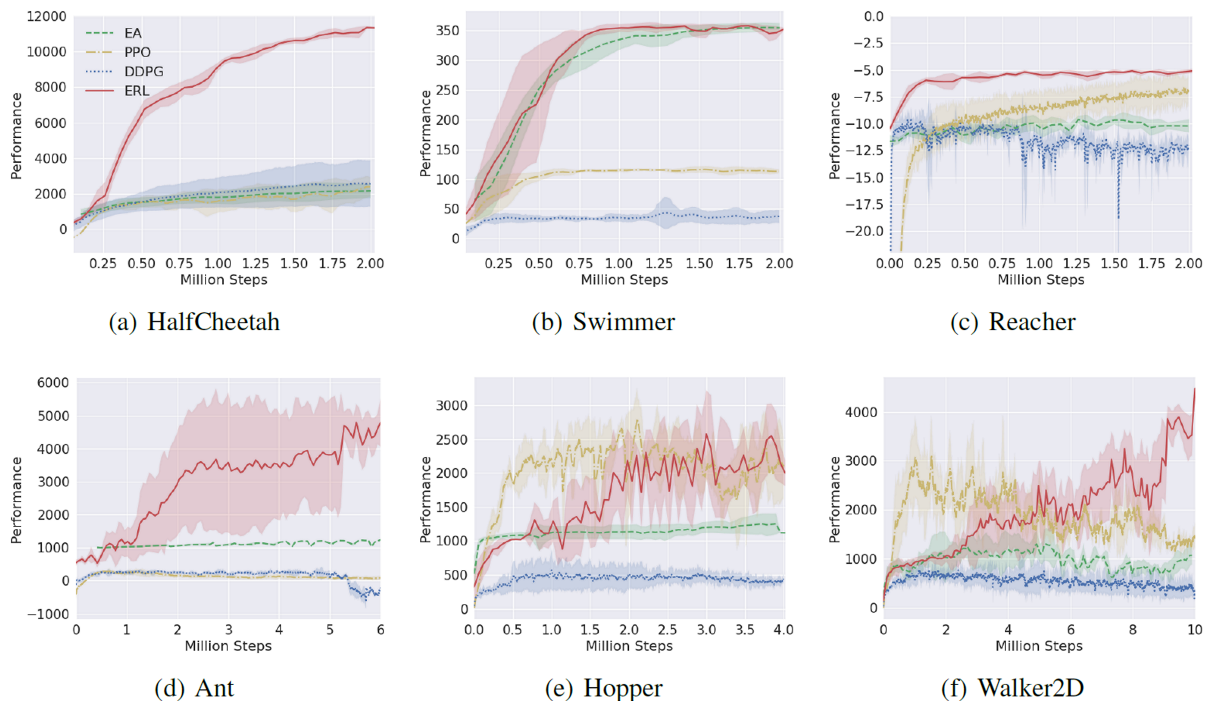


**Figure 3.11:** Performance curves (average reward versus the number of training frames) on MuJoCo continuous control benchmarks. The shaded area corresponds to a standard deviation calculated from 50 trials. Retrieved from [42, 37].

its limitations. They are summarised by Table 3.1 which compiles information from the original papers (cited along with the algorithm name) and the review work done by [37]. All the works listed are very recent. The first version of the ERL paper had been made available in 2018, whereas the last four works have been published after the current study had started.

CERL is developed by the same authors as ERL and it uses a subset of collaborating DDPG actors with shared memory space. In CERL, all actors are injected back into the population [44]. It also includes a simple n-point-based crossover mechanism. Although improving the sample efficiency of ERL in all six MuJoCo tasks, it does not address its main limitations. X-DDPG is another version of ERL that trains several asynchronous DDPG actors but the buffers of the evolutionary agents are separated from the RL buffers. Similarly to ERL, the most recent DDPG actor is injected into the evolutionary population at each synchronisation time [101].

ESAC improves ERL by using the more developed SAC algorithm in place of DDPG and incorporates a modified evolutionary strategy with scaled Gaussian mutation [102]. Despite this, it has the same higher-level architecture as the one shown in Figure 3.10. The gain in performance of ESAC can be partially attributed to the more efficient SAC agent, as a better RL algorithm is indeed expected to inject the actor population with information of higher quality [100, 102].

Directly aiming to improve the stability of the evolutionary algorithm, CEM-RL combines a TD3 agent trained on half the evolutionary population of actors with the CEM algorithm, a simplified variant of the CMA-ES [56].

The PDERL extension directly targets the main concerns of ERL. The standard n-point-base crossover is replaced by a distillation version: both parents join their personal replay buffers and crossover selects the best experiences to fill the buffer of the offspring, before applying behavioural cloning to get a new policy that behaves in accordance with the data in the buffer [43]. For mutation, they adopt a more conservative Gaussian, similar to using a Gauss-Newton method to perform the policy gradient step [37].

Moving away from direct policy learning, EAS shows empirically that evolving the actions and collecting them in an archive is an alternative to evolving policy networks [103]. The algorithm incorporates a TD3 agent and its final performance is similar to other TD3-based frameworks. Moreover, the algorithm was published after the current study started and it has neither received any peer reviews nor other citations. Although interesting, due to its relative recency, it will not be used for the current work.

Via Supe-RL, the authors directly address the safety concerns of the learnt policies. They incorporate a gradient-based mutation operator that should penalise exploratory trajectories that diverge from a safe domain [100].

Lastly, SERL and SPDERL use the ERL and PDERL frameworks but replace the DDPG with a TD3 agent. They both rely on the trained critic network to become a surrogate model of the environment [104]. Using the surrogate makes the evaluation step faster and it can serve as an extension of all the other frameworks presented so far.

**Table 3.1:** Review of the off-policy hybrid frameworks, ordered chronologically based on the date of first publication. Inspired from [37].

| Framework | RL Agent | EA | Actor Injection | Mutation | Crossover | Surrogate Fitness | Dedicated Buffer |
|---|---|---|---|---|---|---|---|
| ERL [42] | DDPG | GA | ✓ | Gaussian | ✗ | ✗ | ✗ |
| CERL [44] | DDPG | GA | ✓ | Gaussian | n-point | ✗ | ✗ |
| CEM-RL [56] | DDPG | CMA | ✗ | ES update | ✗ | ✗ | ✗ |
| PDERL [43] | DDPG | GA | ✓ | Proximal | Distillation | ✗ | ✓ |
| ESAC [102] | SAC | ES | ✓ | AMT Gaussian | n-point | ✗ | ✗ |
| X-DDPG [101] | DDPG | GA | ✓ | ✗ | ✗ | ✗ | ✓ |
| Supe-RL [100] | Rainbow | GA | ✓ | Gradient-based | n-point | ✗ | ✗ |
| EAS [103] | TD3 | GA | ✓ | Gaussian | ✗ | ✗ | ✗ |
| SERL [104] | TD3 | GA | ✓ | Gaussian | n-point | ✓ | ✗ |
| SPDERL [104] | TD3 | GA | ✓ | Proximal | Distillation | ✓ | ✓ |

Most of the papers above use the other algorithm for comparison to illustrate gains in performance or learning stability. Aiming to draw a definitive conclusion on which algorithm performs the best under any given condition is impossible. That is why, by following the argument of majority voting, PDERL and CEM-RL raise above the other frameworks. They both tackle the initially observed limitations of ERL, achieving state-of-the-art training performance on the MuJoCo tasks. Despite CEM-RL usually showing the edge, PDERL only uses the slightly less developed DDPG algorithm [103]. It is expected that, by switching to a TD3 implementation, PDERL can perform as well or even better than CEM-RL.

Last but not least, the ultimate goal is fault-tolerance so the training performance does not provide the entire picture and, for this work, the preservation of safe behaviours is essential. As proved by [100], the mutation operator has to be adjusted to obtain a safe-oriented exploration. Since CEM-RL is based on ES which incorporates mutation only indirectly, the more direct proximal mutation operator used by PDERL is considered more flexible and hence superior. Therefore, the PDERL algorithm will be further described and implemented.

### 3.4.3. Preliminary Method

Proximal Distilled Evolutionary Reinforcement Learning (PDERL) is chosen as a promising framework. Whereas it uses the same structure as ERL and a DDPG agent, it proposes two novel genetic operators. These operators are designed to counteract catastrophic forgetting. Thus, they directly target the instability caused by Gaussian mutation and n-point crossover applied to a direct encoding of NNs.

#### Genetic memory

First, To integrate both operators, PDERL uses an additional set of memory buffers, referred to as the *genetic memory*. As part of the global genetic memory, each actor has its own memory buffer with size $\kappa$ that stores the most recent personal experiences, selected from the main memory buffer. Moreover, each personal buffer can store experiences that span multiple generations.

The experiences stored in the genetic memory are spread through the population by the variational operators. Mutation passes the entire personal buffer whereas crossover fills the offspring's memory by joining half of the most recent experiences collected by its parents.

**Q-filtered distillation crossover**
PDERL selectively merges the behaviours of two elite policies into a new child policy based on their estimated Q-values. In contrast to the n-point crossover of genes, this operator acts in the phenotype space [43]. Essentially, offsprings inherit optimised behaviours from their parents and not just the weights of the networks.

First, consider two distinct parent policies $\mu_x$ and $\mu_y$. An offspring policy $\mu_z$ is created and its buffer $\mathcal{D}_z$ is filled with $\kappa/2$ experience from both parents. its weights $\theta_z$ are randomly initialised with the weights of one of the parents. Then, the child actor is optimised to selectively imitate its parents' actions in the sampled states. This form of *Imitation Learning* can also be seen as a more general type of policy distillation since it aims to imitate or 'distil' the behaviour of the parents into the child policy. In contrast to the model distillation proposed by [105], the size of the resulting model is not smaller than any of the parents as all actors within the population have the same footprint. Furthermore, the crossover operator combines two networks instead of one which introduces the problematic possibility of offspring behaviours diverging from both parents. To account for it, PDERL introduces a cloning loss $L_C$ which the policy $\mu_z$ is trained to minimise over the batch of transitions $B_C$:

$$\begin{aligned} L(C) = & \sum_i^{B_C} \left\| \mu_z(s_i) - \mu_x(s_i) \right\|^2 \mathbb{I}_{Q(s_i,\mu_x(s_i)) > Q(s_i,\mu_y(s_i))} \\ & + \sum_j^{B_C} \left\| \mu_z(s_j) - \mu_y(s_j) \right\|^2 \mathbb{I}_{Q(s_j,\mu_y(s_j)) > Q(s_j,\mu_x(s_j))} \\ & + \frac{1}{B_C} \sum_k^{B_C} \left\| \mu_z(s_k) \right\|^2 \end{aligned} \tag{3.32}$$

The operator $\mathbb{I}$ always selects the policy with a higher Q-value, forcing the children to follow the best-acting parent. The third term on the right-hand side of Equation 3.32 is an $L_2$-norm regularisation that limits the actions from becoming too aggressive. potentially, this term can be updated to directly account for unsafe actions.

Lastly, one still has to decide which elites are going to generate the offspring. In PDERL, it is argued that the best selection strategy is to pair individuals based on the distance between their expected actions. Equation 3.33 defines the metric as the expected Euclidean distance between the actions taken by the actors over a batch of states:

$$d(\mu_x, \mu_y) = \mathbb{E}_{\mathbf{s}\sim\rho_x}\left[\left\|\mu_x(\mathbf{s}) - \mu_y(\mathbf{s})\right\|^2\right] + \mathbb{E}_{\mathbf{s}\sim\rho_y}\left[\left\|\mu_x(\mathbf{s}) - \mu_y(\mathbf{s})\right\|^2\right] \tag{3.33}$$

With the $\rho_{x,y}$ denoting the state-visitation probability distribution of each parent actor. In practice, the expectation operator is approximated over the states sampled from the genetic memories of the individuals.

The probability of selecting a pair increases with the distance between them. Therefore, it is expected that a distance-based selection strategy incentives novelty. Consequently, since some fit individuals might not be selected, the short-term performance of the population might suffer which is another trade-off between exploitation and exploration.

**Proximal mutation**
ERL algorithms mutate the policies to explore the search space. Unfortunately, the risks of aggressive mutations are two-folded: they might lead to unsafe actions being taken and they can destabilise learning. If a reasonably well-performing policy is randomly perturbed, it might become impossible for it converges again.

To limit the destructive effects, PDERL implements a safe mutation operator inspired from [106]. To understand how they aim to achieve this, consider the actor to be mutated as $\mu_x$ parametrised by $\theta_x$. The mutation operator samples a batch of states $B_M$ from the genetic memory of the actor. By summing the

**Algorithm: Distillation Crossover**

```
    Input: selected parent policies $\mu_x$, $\mu_y$ with buffers $\mathcal{D}_x$,$\mathcal{D}_y$
    Output: Child policy $\mu_z$ with memory $\mathcal{D}_z$
 1: Add latest $\frac{\kappa}{2}$ transitions from $\mathcal{D}_x$ to $\mathcal{D}_z$
 2: Add latest $\frac{\kappa}{2}$ transitions from $\mathcal{D}_y$ to $\mathcal{D}_z$
 3: Shuffle $\mathcal{D}_z$
 4: Initialise the weights $\theta_z$ with the weights of one of the parents at random
 5: for $e$ in $range(total\_epochs)$ do
 6:     for $iter$ in $range(\kappa/B_C)$ do
 7:         Sample transitions batch of size $B_C$ from $\mathcal{D}_z$
 8:         Optimise $\theta_z$ to minimise $\mathcal{L}_C$ using Adam
 9:     end for
10: end for
```

gradient of each dimension of the output action over these states, the sensitivity of the policy is computed with Equation 3.34:

$$\boldsymbol{s} := \sqrt{\sum_{k}^{|\mathcal{A}|} \left( \sum_{i}^{B_M} \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}} \left(s_i\right)_k \right)^2} \tag{3.34}$$

Where $\mathcal{A}$ represents the set of all possible actions. Then, the sensitivity is used to scale the zero-mean Gaussian perturbation, the policy parameters being updated according to Equation 3.35:

$$\theta' \leftarrow \theta + \frac{\epsilon}{s} \quad \text{with} \quad \epsilon \sim \mathcal{N}(0, \sigma \boldsymbol{I}) \tag{3.35}$$

With $\sigma$ being the mutation magnitude. The higher the magnitude of the policy gradient with respect to a parameter, the smaller the mutation step becomes. Thus, sensitive parameters are perturbed less. After the scaled update, the resulting policy $\mu_{x'}$ has its parameters $\theta'$ within the proximity of its predecessor. Hence, it can be assumed that his behaviour will not diverge too much either.

For this mutation operator, safety strictly refers to the ability to converge to a policy during learning. Oder aspects of safety are relevant, for example, [107] lists multiple works that offer a negative reward signal proportional to the variance. These alternatives will be investigated in future work.

Figure 3.12 shows a high-level concept of how the genetic memory, distillation crossover and proximal mutation add to the ERL-based algorithm from Section 3.4.1.

**Comparison and discussion**

Similar to ERL, PDERL had been bench-marked on the same MuJoCo control tasks as ERL. The PDERL plots from Figure 3.13 mirror the ones shown by Figure 3.11 apart from the Reacher task (plot (c) of Figure 3.11) which has been dropped. To showcase its performance, it is compared against state-of-the-art algorithms: TD3, PPO and ERL. Moreover, two ablated versions of the framework are considered: PERL only involves the proximal mutation and DERL performs the simple Gaussian mutation with distillation crossover.

The works presented in this study usually rank methods based on the final performance of the best-performing actor, the champion. For the current study, this metric would not draw a complete picture. By taking safety into account, an estimation of the **worst-case performance** shall be used. The methods will be compared by computing the final reward minus half the standard deviation for each evaluation case.

PDERL consistently scores better than the other algorithms. For the more simple environments, such as Hopper and Swimmer, the performance differences to its ablated versions are within the performance gap defined in Chapter 3.2 so the results can be considered similar. The more complex tasks (Ant and Walker) show a better distinguishable hierarchy. In these two tasks, the worst-case performance of PDERL raises above the average performance of almost all the other algorithms. Whereas TD3 matches it on Walker2d, it is significantly over-scored in the Swimmer environment.

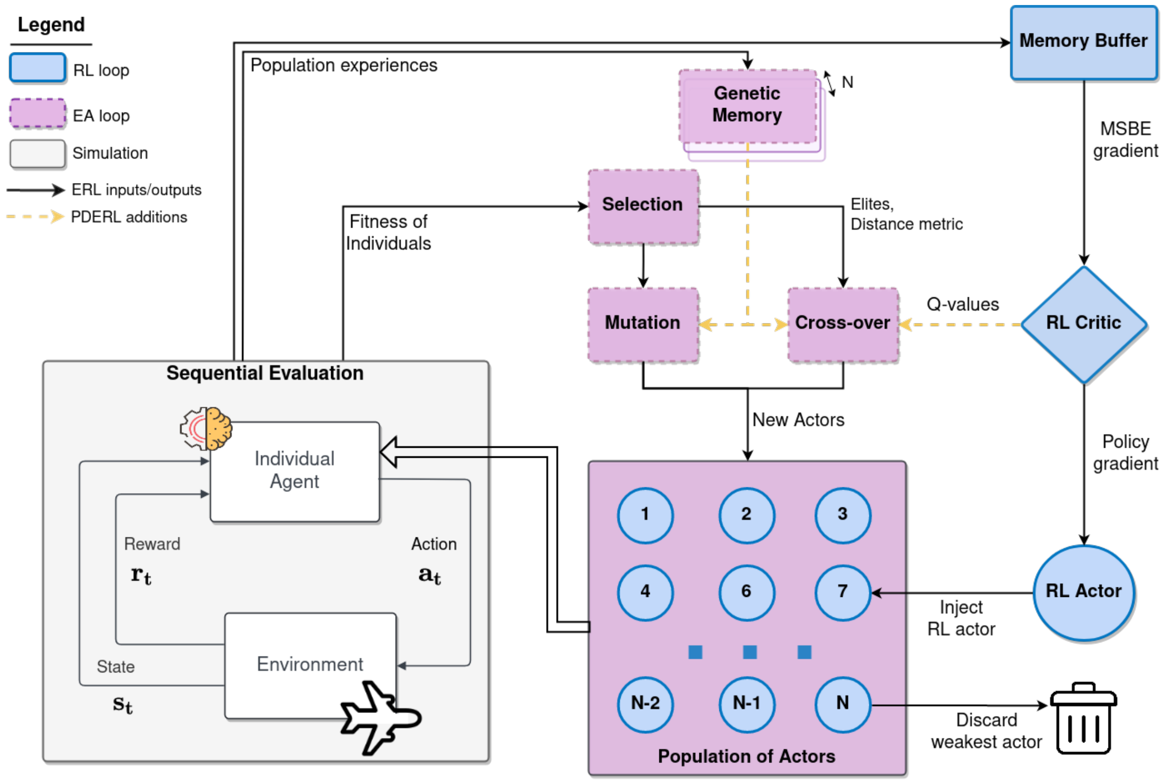


**Figure 3.12:** High-level visualisation of Proximal Distilled Evolutionary Reinforcement Learning (PDERL), highlighting the EA's population-based learning (purple) with DRL's gradient-based optimisation (blue). The interrupted yellow line highlight the additional input/output connection that PDERL uses on top of the standard ERL. Adapted from [43] based on [37].

It is worth mentioning, that TD3 matches PDERL on all environments apart from Swimmer. According to [56], the Swimmer task provides deceptive gradient information which makes any pure RL framework fail to converge. Out of all MuJoCo tasks, Swimmer is skipped by the papers providing the DRL state-of-the-art [12, 13, 14].

Also, whereas both DERL and PERL perform better than ERL, PDERL is always at least as good as the best out of these two. Both the distillation crossover and proximal mutation improve performance.

Furthermore, by looking at the difference between PERL and ERL, the authors prove empirically that the policies mutated by additive Gaussian noise completely diverge from the behaviour of the parent. These sudden and drastic changes in behaviour are a sign of *catastrophic forgetting*, proving that the concerns about the undesirable effects of Gaussian mutation are not to be ignored.

PDERL consistently achieves more stable performance. Its variance is lower than other state-of-the-art algorithms with comparable rewards. This excludes PPO which generally converges towards the lowest variance but its average performance is at best $30\%$ lower. The authors again propose that the proximal mutation operator has the largest effect on the improved robustness of the converged behaviour. As safety is of paramount importance, robust behaviour is more than desirable.

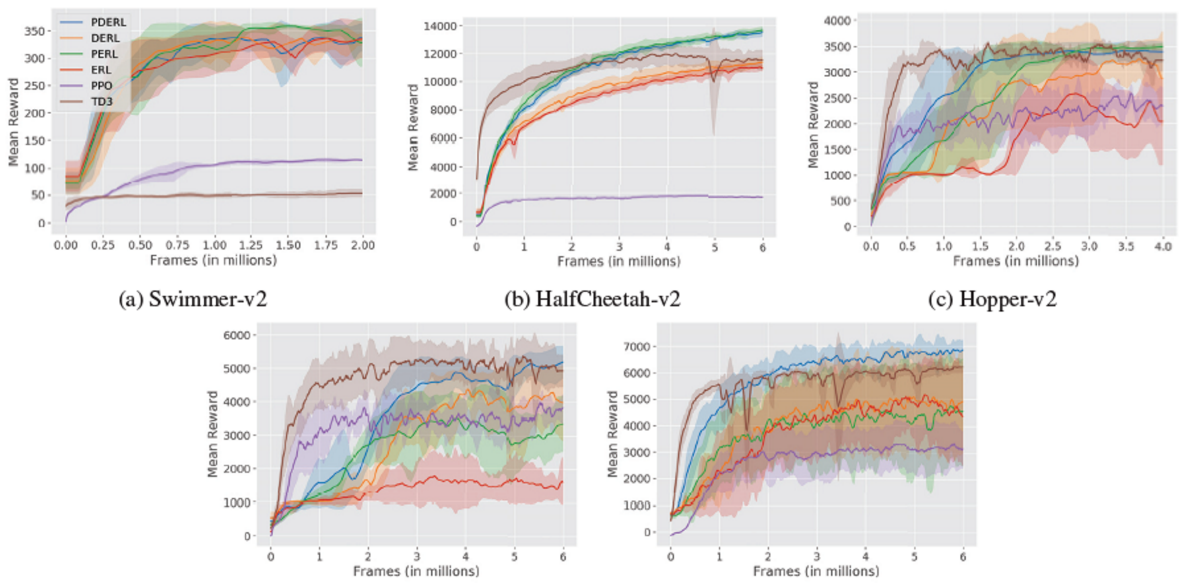


**Figure 3.13:** Performance curves of PDERL on MuJoCo continuous control benchmarks. The shaded area corresponds to a standard deviation calculated from 30 trials. Retrieved from [43].

### 3.4.4. Conclusion

ERL trains in an evolutionary loop a population of control policies, periodically infusing it with new actors trained by gradient-descent on the global experiences. Whereas the algorithm manages to combine the best of both worlds, its limitations surface and multiple frameworks aim to correct them. By identifying these frameworks, sub-question **Met-2.c** has been covered.

PDERL achieves high average performance in all control tasks used within the literature. More than its performance compared to other state-of-the-art algorithms, the learnt controllers benefit from a low variance over evaluation runs, showing robustness to changes in the initial conditions. This is achieved by specifically reducing the chance of catastrophic forgetting via proximal mutation and distillation crossover. The literature showed that this improves learning stability and worst-case performance. Moreover, the effects of both operators are directly traceable which eases future exploration of possible improvements. Due to its robust behaviour and the opportunity to easily upgrade it, it has been selected as the base framework for the task at hand.

Crucially for the problem at hand, its fault-tolerance performance has yet not been investigated in the literature. The extent to which the PDERL population of actors can generalise to an unforeseen circumstance and remain robust to failures is to be tested. This will be the goal of the next chapter's preliminary analysis.

# 4 Preliminary Results

This chapter concerns itself with describing the implementation and testing of a preliminary hybrid ERL framework tasked to solve a simple continuous control toy problem. It aims to verify the feasibility of adopting a hybrid ERL algorithm for fault-tolerant control, based on the insights presented in Section 3.4. Whereas a set of performance comparisons will be shown, the objective is to empirically verify the feasibility of adopting such hybrid methods to solve the task at hand. Thus, the current chapter contributes towards the research goal, answering the final question within the Method Feasibility set, namely, question **Met-3**. Last but not least, exhaustive comparisons and trade-offs meant to identify the overall best method reach beyond the scope of this work.

The policies are trained offline and then tested against two failure cases. AS PDERL uses a DDPG for the reinforcement learning agent, the two were compared during training, in a comparable manner to what is already available in the literature surveyed by Section 3.4. Also, the CMA-ME quality-diversity algorithm developed by [87] was reproduced to have the behaviours resulting from PDERL compared against its solution.

First, a simple continuous control environment is defined by Section 4.1 in the form of LunarLander. Then, the offline training of PDERL, DDPG and QD is presented in Section 4.2. Then, Section 4.3 shows the behaviours that the agent exhibits by following the learnt policies. Finally, Section 4.4 tests the performance of the offline trained algorithms on a faulty.

Both training and testing are performed in Python using the PyTorch[1] library. Following the descriptions of the previous sections, the algorithms used are taken from the original papers' GitHub repositories, which are made publically available. For the QD algorithm, the core of the original repository has been made available in the form of an open-source Python library called Pyribs [2] As the library benefits from more extensive tests and better visualisation tools, it has been used for QD implementation.

## 4.1. Testing Environment

The algorithms are implemented on a continuous control benchmark, the *LunarLander* environment offered within OpenAI Gym[3]. This environment is part of the testing toolbox for optimal control and RL algorithms, being inspired by the set of classic arcade lunar landing games from the 70s. This environment was selected because it provides a continuous reward signal while being more complex than the classic control tasks such as the inverted pendulum, the double pendulum or the cart-pole system. Also, it is more simple than the MuJoCo tasks. Hence, it reduces the run-time to around one hour on a personal machine in comparison to at least four hours needed for any MuJoCo environment.

The agent is a lander with three thrusters which are controlled to guide its landing on the lunar surface. Figure 4.1 provides a visualisation of the environment, agent and the corresponding coordinate frame used to define its state.

At the beginning of each trial, the lander spawns at a random location at top of the map. Its initial speed is zero and its orientation is vertical. The relief of the lunar surface is also randomised. At each time step

[1]`https://pytorch.org/docs/stable/index.html` [Visited on 28/05/2022]

[2]`https://pyribs.org/` [Visited on 25/05/2022].

[3]`https://gym.openai.com/envs/LunarLander-v2/` [Visited on 10/04/2022]

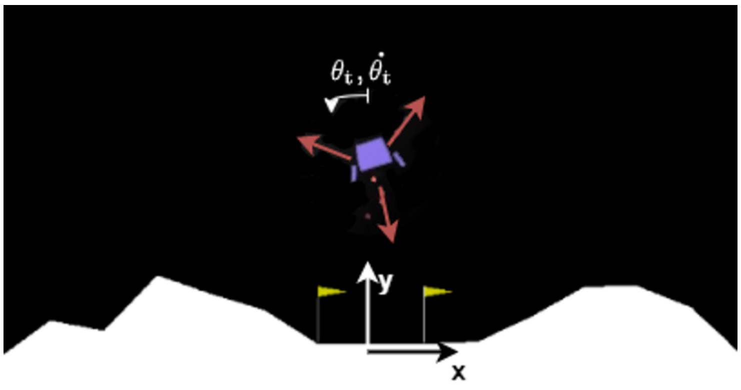


**Figure 4.1:** *LunarLander* environment and its coordinate frame.

during the trial, the agent can only use its three thrusters to control the landing. The goal is to land with zero speed on the landing pad marked by the two yellow flags. The task has a finite time horizon as the episode terminates after 1000 steps, irrespective of the outcome. Lastly, the environment is deterministic since there is no stochasticity in the effects of actions or the rewards obtained.

**State and action space**
In this environment located at coordinates (0, 0), as illustrated in Figure 4.1. The entire information with regards to its state is known from the observations available at each time step. Thus, the state is fully observable.

$$\mathbf{s_t} = \left[x_t, y_t, \dot{x}_t, \dot{y}_t, \theta_t, \dot{\theta}_t, ll_{touch}, rl_{touch}\right]^\top \quad \mathbf{a_t} = \left[T_t^{main}, T_t^{lat}\right]^\top \tag{4.1}$$

The lander position is defined relative to the centre of the landing pad, with $x$ pointed towards the left-hand side of the environment and $y$ pointing upwards. The lander's orientation at time $t$ is $\theta_t$, taken as counter-clockwise positive from the y-axis.

The first six states are continuous whereas the last two are discrete with $ll_{touch}$ and $rl_{touch}$ Booleans that become $1$ once the left or right landing gear touches the lunar surface. The action space is dimensional, each component taking the following values:

- $a_0 = T^{main} \in [-1.0, 1.0]$: the thrust value of the main engine (middle one). It is operational for $a_0 > 0$, otherwise, it is set to idle.
- $a_1 = T^{lat} \in [-1.0, 1.0]$: controls the lateral engines. The right thruster fires for $a_1 \leq -0.5$ whereas the left one when $a_1 \geq 0.5$. Otherwise, they are both set to idle.

**Reward structure**
The reward for moving from the top of the screen to the landing pad with zero speed is awarded between 100 and 140 points. If the lander moves away from the landing pad it loses the same reward as moving the same distance toward the pad. The episode receives additional -100 or +100 points for crashing or landing, respectively.

At each step, using the main thruster (i.e., the one in the middle) burns fuel so it returns $-0.3$ reward points. Finally, a reward of +10 points can be offered for landing with one leg on the ground but this option has not been used in the current tests to increase the importance of the continuously provided reward.

Empirical evidence shows that an episode score of 200 or more can be considered a solution since it mimics human-level performance on the task.

**Behavioural characteristics**
The QD algorithm requires the behavioural space to be defined by the user. As there are many ways to successfully land on the surface, the span behaviours can help the user understand how the system

performs the task. By checking the behaviours, one can realise what is the outcome of following certain policies and not only how successful they are.

For the current system, the behavioural characteristics are defined according to Equation 4.2 as the horizontal coordinate and the vertical velocity at touchdown. If the episode ends before touchdown, the final $x$ and $\dot{y}$ values are taken.

$$\mathbf{b} = [x_{impact}, \dot{y}_{impact}]^{\top} \tag{4.2}$$

In spite of the current implementation of PDERL not using these behavioural characteristics, the same values can be outputted by the algorithm. By doing so, one can compare the behaviours learnt by PDERL and the ones optimised by QD.

## 4.2. Offline Training

Each of the algorithms has been trained offline for $0.8 \cdot 10^6$ frames each. This proved to be enough for both of them to converge. The criterion chosen for convergence was that no performance metric was changing by more than $2.5$% (half the performance gap) between evaluations. The metrics recorded are the average reward of the champion, average population reward, and policy loss ( the Q-value).

### Set of hyperparameters

Where applicable, the same hyperparameters have been used. For DDPG, the optimal values for the LunarLander environment were taken from the StableBaseline 3[4]. these values are expected to be extensively tuned for this particular environment. The rest of the parameters were taken directly from the original implementations [12]. For PDERL, the RL part shares the same parameters with DDPG whereas settings of the evolutionary were also taken from the original paper [43]. Table 4.1 summarises the used parameters.

**Table 4.1:** Hyperparameters for the DDPG and PDERL algorithms.

| **DDPG** (both) | | **GA** (only PDERL) | |
|---|---|---|---|
| Learning rate | 0.7 | Population size | 30 |
| Discount factor | 0.98 | Elite fraction | 0.2 |
| Batch size | 128 | Mutation magnitude | 0.05 |
| Buffer size | 200,000 | Mutation probability | 0.9 |
| Polyak rate | 0.001 | Population evaluations | 3 |
| Delayed start (frames) | 10,000 | Synchronisation rate | 1 |
| Actor hidden layers | 2 | Genetic memory size | 8,000 |
| Critic hidden layers | 2 | Champion evaluations | 50 |
| Actor hidden sizes | (300, 400) | | |
| Critic hidden size | (200,300) | | |
| Non-linear activation | ELU | | |

### Training curves

Figure 4.2 shows the learning performance of both algorithms against the number of frames passed. Since both algorithms use an RL loop, the amount of information learnt is directly dependent on the number of frames experienced during taring.

Whereas DDPG only has one policy, the population of PDERL is depicted by taking the reward of the population champion (solid line) and the average over the entire population (interrupted line). The curves are similar to the performance figures from Sections 3.4.1 and 3.4.3, apart from the population average which is not present in previous plots.

[4]`https://github.com/DLR-RM/rl-baselines3-zoo/blob/master/hyperparams/ddpg.yml` [Visited on 20/05/2022]

One can see that both algorithms solve the task, converging in high rewards. PDERL has the edge with a final reward higher by $5.6\%$, passing the performance gap of $5\%$ set in Chapter 3.2. Its sample efficiency is also higher, as the best-behaving actor consistently reaches rewards higher than 200 within the first $180 \cdot 10^3$ frames. Moreover, the average over the population matches the performance of DPDG within $1\%$ difference. Despite this, the RL algorithm shows high variation over the runs, deviating from the average by more than $25\%$.

Overall, PDERL is more stable. The parts of the plot where the standard deviation suddenly spikes up are caused by changes in the incumbent champion. The champion is selected by seeing which policy behaves the best over 3 evaluations of the population. Thus, sometimes the best individual is not correctly identified. The frequency of evaluating the champion is much higher, currently being set to 50.

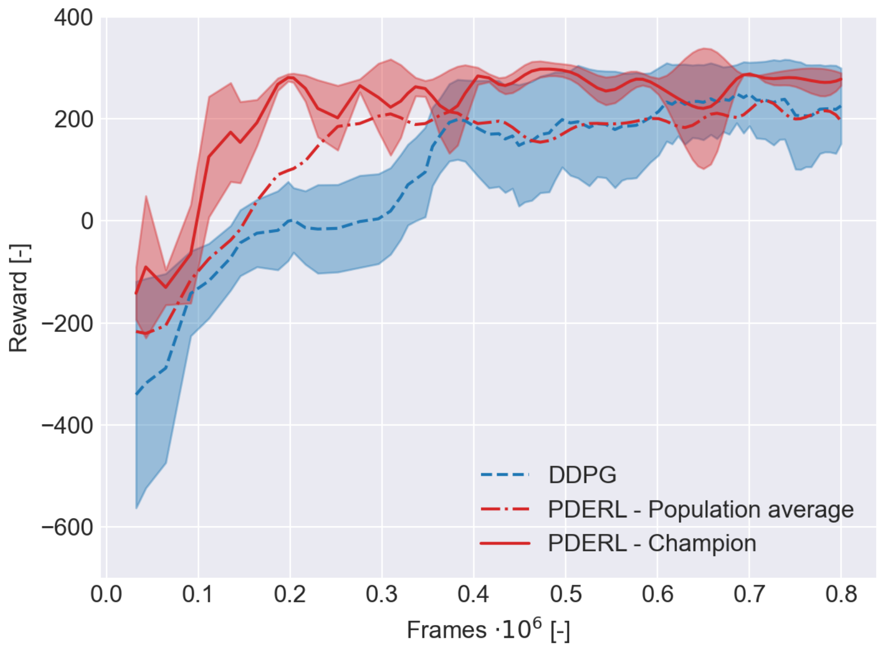


**Figure 4.2:** Offline training performance of DDPG and ERL algorithms. The shaded areas correspond to the standard deviation taken over 50 evaluations of the champion. The first 100 frames are skipped for better scaling of the reward axis.

**Training QD**

A direct comparison between QD and the other two training processes cannot be done. A generation in EA does not correspond to one epoch of gradient descent. As it is based on the CMA-ES evolutionary algorithm, this framework does not learn directly from the individual frames but from the completed episodes. Because of this, one should proceed with caution. Even for a linear policy, QD requires a much higher number of episodes to illuminate the entire archive of behaviours. By adapting the settings from the original paper to the LunarLander task, the number of episodes played during training is $50000$. This translates to an upper limit of $5 \cdot 10^8$ frames but in practice, the number varies depending on how fast the agent manages to learn. The required number of frames is significantly higher than for the other two algorithms. This might be expected since, even though the policies are linear (given by Equation 4.3), QD learns a total of $2500$ from them.

$$a = W \cdot s \tag{4.3}$$

The model footprint of each of these policies is $16$, corresponding to the size of the weight matrix $W$. This is a few orders of magnitude lower than the model footprint of the PDERL and DDPG actors. They use multi-layered neural networks, and each of them contains more than $120000$ learnable parameters.

Thus, training non-linear policies and stopping the QD algorithm at $10^6$ frames will result in non-converged behaviours. Moreover, the CMA-ME uses the emitters to fill the archive map iteratively with solutions increasingly better in terms of quality and diversity. It does not only evaluate an average episodic reward and it cannot estimate its deviation. These differences make the QD equivalent of a performance curve not comparable to the other two algorithms.

Nevertheless, it has been decided to keep the linear policies as, for the simple system at hand, they still result in informative behaviours. Section4.4 will investigate the robustness of these policies against system failures.

The QD implementation has no hyperparameters to tune apart from the number of iterations which was taken as $400$ as the original paper recommends a number in the order of $10^2$ to $10^3$ [87]. Lastly, the *improvement emitter* was used as it offered the most balance between quality and novelty (refer back to Section 3.3.2).

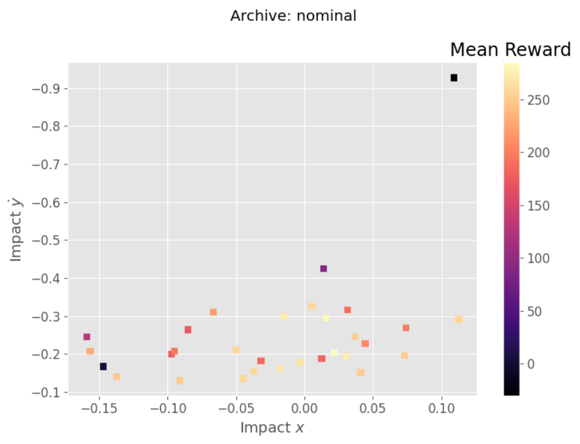
## 4.3. Repertoire of Behaviours

The QD optimisation results in an archive of elites which can be plotted along with the bi-dimensional behavioural space. The same plot is built for the behaviours resulting from following the population of policies trained with PDERL. One policy returns one behaviour which becomes one point on the map. Figure 4.3 shows the two maps with the colour corresponding to the total reward of each policy.

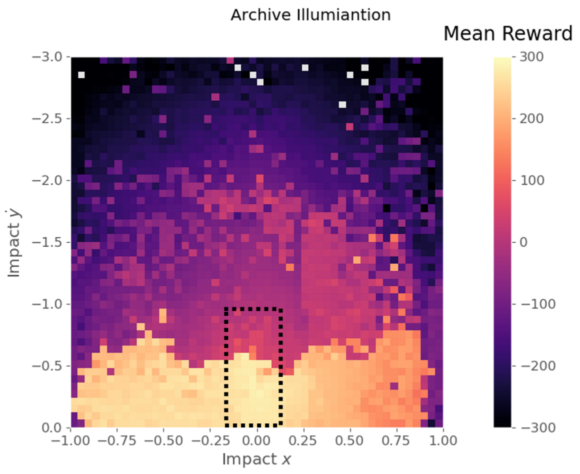
Both maps show the benefit of policies that result in low vertical speed at impact. This insight is expected because a higher is offered when the total final speed is zero. Moreover, touching-down closer to the middle of the landing pad is useful but not required. When the lander touches the ground further away from the pad, it can use its lateral thrusts to quickly move along the surface for a better reward. Unfortunately, such behaviours are highly sensitive to the shape of the surface. If it is too bumpy, they might not be able to arrive on the pad before the episode finishes.

On one hand, the QD plot offers insights concerning the global performance of behaviours. For the nominal task, one can rather clearly identify the region within the behavioural space that corresponds to higher performance.

On the other hand, PDERL only trains 30 policies so its behavioural map is sparse and it is fully contained by the QD archive. After consulting the plots, one might argue that the ERL-inspired algorithm samples a few of the best-behaving policies. Despite its reduced coverage, the quality of the PDERL solutions is generally higher.



**(a)** PDERL behavioural archive.



**(b)** QD behavioural archive.

**Figure 4.3:** Archive of elites plotted on the behavioural map. The reward and behaviour characteristics are scaled accordingly, with the vertical axis being inverted. The left-hand map is contained by the highlighted rectangular box from the right-hand plot.

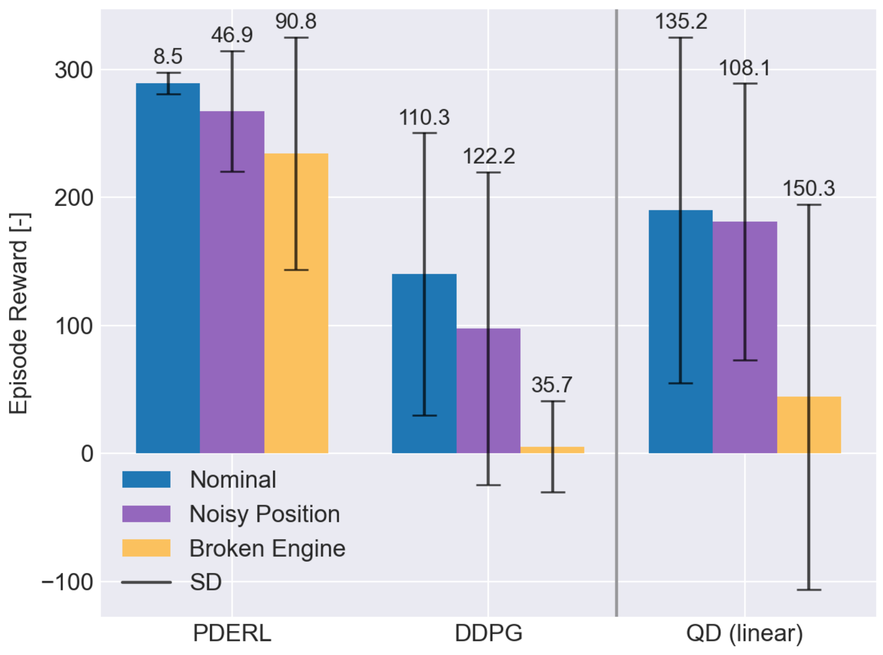


**Figure 4.4:** The average performance and deviation for the learned control policies tested on the nominal systems and the faulty system. The vertical bars correspond to one standard deviation. QD is shown separately since it is not directly comparable but still informative.

## 4.4. Robustness to Faults

To investigate the robustness of the ERL framework against system faults, the nominal LunarLander environment is altered. Taking inspiration from Chapter 3.1, the following cases are considered:

**F1. Faulty sensor recordings:** zero-mean Gaussian noise is added to observed (x,y) coordinates of the lander.

**F2. Actuator failure:** the main thruster is 'broken' by clipping its effective thrust to $75\%$ of the maximum. The fuel consumption stays the same, i.e, there are no changes to the reward function.

Then, the offline trained populations can now be subjected to the task of controlling the damaged systems. For each fault case, the environment is reset for $100$ trials. The results are plotted below in Figure 4.4.

The population evolved through PDERL achieves the highest performance out of all three frameworks and proved robust against both fault cases. Over both faults, it has a drop in the average return of less than $18\%$ and a decrease in its worst-case performance (average performance reduced by the deviation) of $33\%$. Its worst-case reward in the presence of fault F2 even exceeds by $35\%$ the average performance obtained by the RL algorithm on the nominal system.

Despite showing comparable training performance, DDPG is highly sensitive to any changes done to the environment or agent. This translate into a standard deviation of more than $110$ for the nominal case. DDPG can adapt to neither of the faults as its average return decreases to below $100$ and the reward variation remains significant.

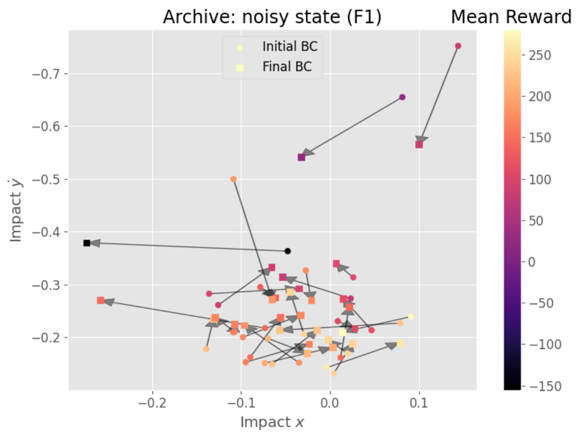
The QD algorithm achieves a higher average reward for the DDPG agent for the nominal system but the entire archive of controllers is heavily affected by the F2 fault. As QD only learns linear policies, it is expected to be more sensitive to drastic changes in the controlled system, such as the broken engine. A noisy observation of its position affects the average reward by less than $10\%$. Despite this, the standard deviation increases from the nominal case by more than $53\%$ for F1 and up to $113\%$ for F2. Similar to the DDPG agent, QD is not able to solve the task under the clipped thruster conditions.

**Changes in behaviour**

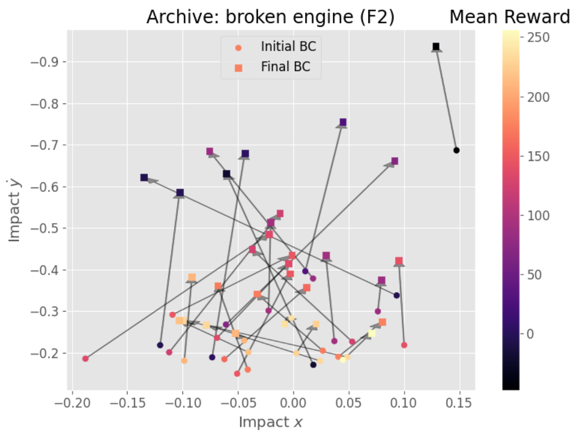
From all three groups, it is clear that adding noise to the observed position is less problematic to deal with than having the main actuator clipped. A potential explanation comes from the physics of the environment and the relative severity of the faults. For all frameworks, a noisy state might mimic the effects of a stochastic policy. It degrades the average performance but it improves the quality of a few selected solutions [5]. That translates into a lower average reward but a higher variation. At the same time, by clipping the main engine, the lander falls faster.

The updated behavioural maps from Figure 4.5 depict the effect faults have on the agent's behaviour. The F1 fault seemingly shifts all behaviours in a random direction by amounts roughly correlated with change in reward. Interestingly, F2 moves all behaviours towards higher $\dot{y}_{impact}$ whereas the other BC is not significantly affected. It seems that the majority of policies cannot compensate for the less efficient thruster. Despite this, the most successful one (coloured bright yellow in the figure) does not change its behaviour significantly. This shows that a policy robust against this fault would keep the engine throttle at lower levels and thus be affected less by the clipping function. For both plots, the region corresponding to high rewards seems to change less in the behavioural space (i.e., shorter arrow lengths). This aspect will be investigated more thoroughly in future work.

Two key insights can be drawn from looking at how the behaviours are affected. First, the behavioural space is directly dependent on the system's configuration. Once a fault is inflected, the behavioural changes. Second, the behaviours that correspond to the highest reward have less sensitive characteristics. In Figure 4.5, the behaviours that change the least have the higher rewards. Therefore, robust behaviour will maintain its performance. This offers some intuition into why the repertoire-learning methods (see Section 3.3.3) can adapt to faults online by estimating how much the resulting behaviours would change.



**(a)** Behaviours under noisy position estimation.



**(b)** Behaviours with the main engine clipped.

**Figure 4.5:** Behaviours obtained by following the trained policies. The reward is scaled accordingly and the arrows point from the nominal original behaviour characteristics (circles) to the current ones (squares).

## 4.5. Conclusion of Preliminary Analysis

The chapter tested the feasibility of using a hybrid Evolutionary Reinforcement Learning framework for fault-tolerant continuous control. Based on the literature surveyed in the previous chapters, Proximal Distilled Evolutionary Reinforcement Learning was chosen as a candidate.

The benchmark used for implementation is the LunarLander, an open-source continuous control environment. On one hand, this environment was selected because it offers a continuous reward signal. On the other hand, it takes less time to train than in the industry-popular MuJoCo environments, making of-

[5]Stochastic policy algorithms benefit from the added exploration during training

fline training feasible for the computational resources at hand. The task is still complex enough to provide a clear ranking between the algorithms at hand and allows for several faults to be inflected.

To compare the robustness, a worst-case scenario metric was defined as the average reward minus half the standard deviation estimate. Both DDPG and PDERL were trained for the same number of frames and their performance was evaluated on systems in the presence of faults. Two cases were considered, inspired by the faults indefinite in Section 3.1, namely a noisy position observation and a clipped thruster. Evaluated on the two faulty cases, PDERL showed a worst-case reward higher than the average DDPG reward by $42\%$ and $25\%$, respectively. Thus, the fault-tolerant capabilities of the hybrid method have been empirically shown, answering questions **Met-2.d** and **Met-3**. Moreover, whereas linear policies can control the agent under nominal conditions, the chapter shows that there is a performance benefit from using non-linear function approximators at the expense of computational time.

Lastly, a quality-diversity algorithm, namely the CMA-ME, was trained in the same environment but under different conditions. The algorithm generated an archive of linear policies that showed highly diverse behaviours in solving the task. For this task, the behaviour was characterised by the horizontal coordinate and vertical speed at impact time. From the resulting behavioural map, one could identify sectors of behaviours that would, under nominal conditions, perform better than the rest. The population of actors trained by PDERL could be seen as a subset of these well-behaving policies.

The behavioural maps were also used to identify the possible effects of faults. The final results presented in Section 4.4 showed empirically that the behaviours resulting from the best-performing policies are less affected by the inflected faults. This result opens up the possibility of using the change in behavioural characteristics as a metric to select the most robust policies. The characteristics are system-specific, so their potential utility for online adaptation will be investigated in future work.

# Part III

# Additional Results

# 5 Hyperparameter Tuning

The hyperparameters are tuned using the Gaussian Process search routine offered by Weights&Biases[1] which took the average population return as the metric to optimise for. The episodic return for each actor agent is averaged over 5 evaluations from the same seed to account for the variation in return.

For each agent configuration, 10 sets of parameters are swept through. The set of parameters used to tune the TD3 agent is visible in Figure 5.1 whereas Figures 5.2 and 5.3 show the tuning results for the two SERL populations of 10 and 50 actors respectively. The bounds of each vertical axes show the range corresponding to the hyperparameter. The chosen set is printed within the Methodology section from Part I.

The present analysis provides evidence that hyperparameter tuning remains a necessary step of ERL development. Looking at the two figures, one might realise that, although none of the parameters provides a definitive effect on the learning performance, specific combinations seem beneficial. For both frameworks, a large learning rate is more prone to diverge whereas a small one slows down learning. The same is true for the hidden size as large neural networks can learn more complex behaviours but with a lower convergence rate. For SERL, combining a large exploratory noise with drastic mutations could increase the likelihood of catastrophic forgetting. Thus, it seems beneficial for learning performance to use a balanced combination.

The preliminary analysis showed that $N$, the size of the genetic population, has the most significant effects and should be considered separately. Hence, the two sweeps for SERL. Notably, it also impacts the control policy smoothness, as it has been discussed in Part I. Subsequently, the learning rate, the size of the memory buffer and the maximum number of training frames have to be adjusted proportionally to the number of trained actors. During one generation, more actors would collect more experiences so the buffer fills up more rapidly. To prevent the TD3 actor and critics from diverging, the gradient updates have to be slowed down. Despite this, increasing the population size seems to improve the average over the entire population but a statistical analysis to verify this claim has been left as a future recommendation.

Most of the hyperparameters trace back to the TD3 agent. Future research should aim at transferring parameters first tuned on TD3 (which is a simpler task) and then scale them for SERL. It is also worth considering that for larger population sizes the rate at which the TD3 actor is selected among the elites is slimmer. Given that the elitism information dissipates more through the population, in such cases, it becomes less relevant how well it behaves. Potentially, in the case of large population sizes, the TD3 hyperparameters become less relevant for the final performance whereas they can still affect the sample efficiency.

Lastly, to increase the accuracy of the analysis, multiple sweeps over shorter ranges should be considered.

[1]Weights & Biases Documentation: `https://docs.wandb.ai/guides/sweeps`

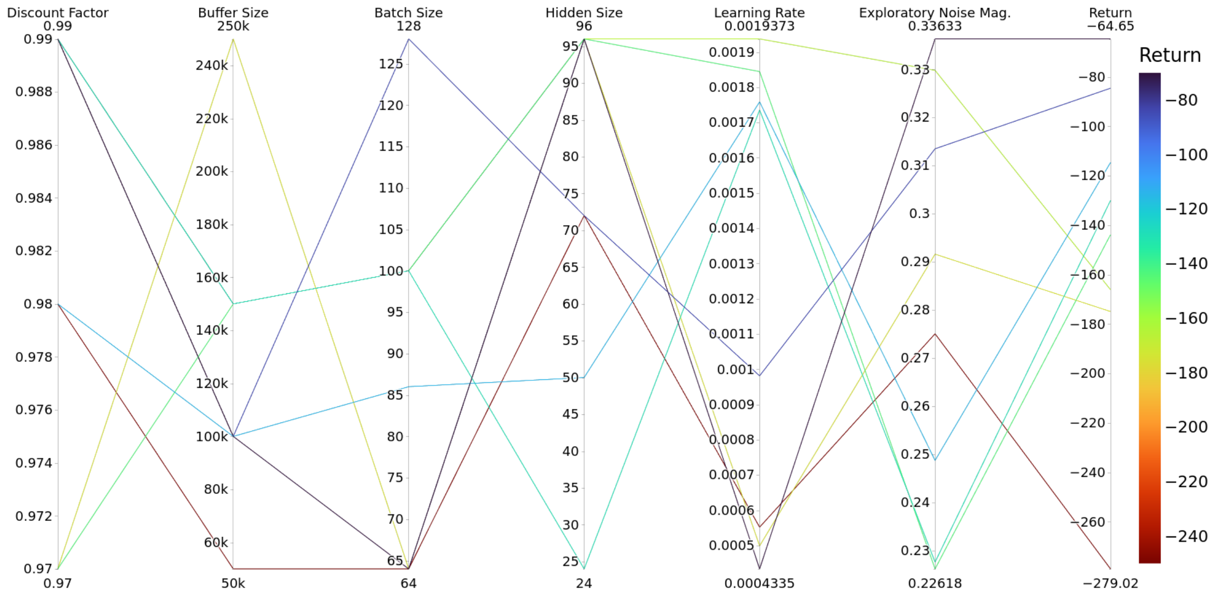


**Figure 5.1:** Parallel coordinates plot obtained from the sweep of selected TD3 hyperparameters. Each configuration has been trained for $0.8 \cdot 10^6$ frames.

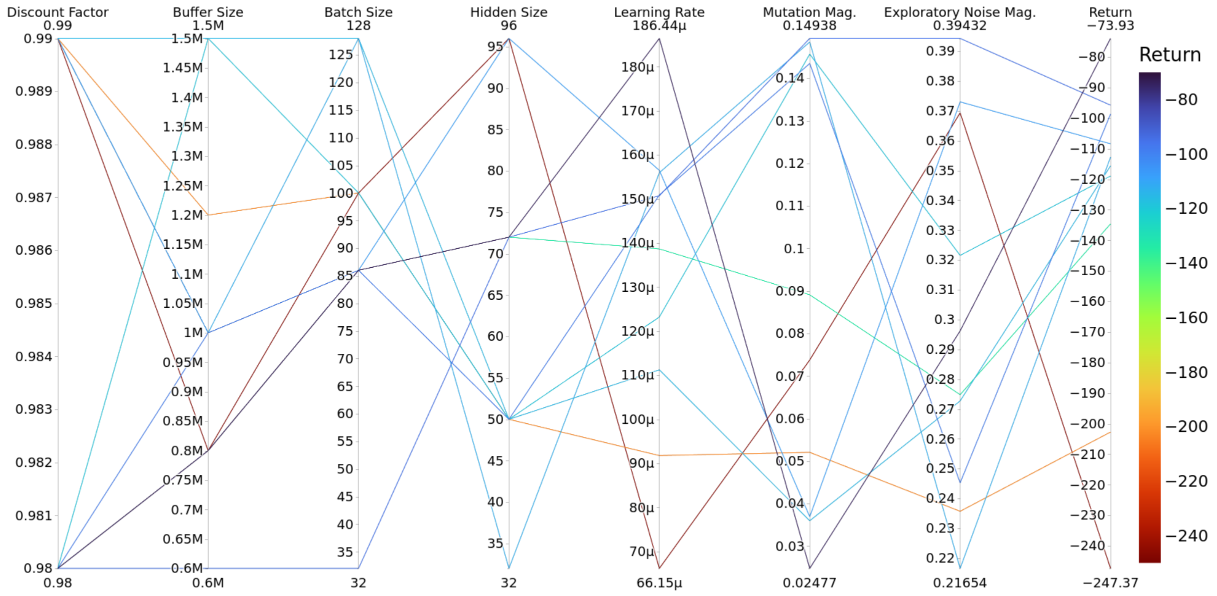


**Figure 5.2:** Parallel coordinates plot obtained from the sweep of selected SERL(10) hyperparameters. Each configuration has been trained for $10^6$ frames.

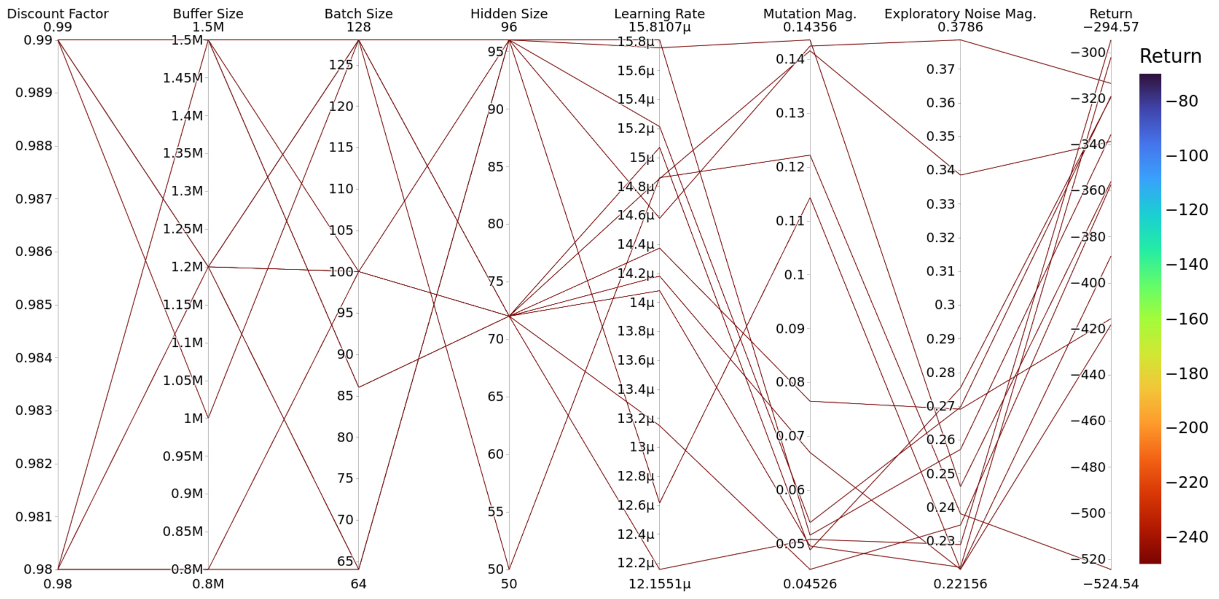


**Figure 5.3:** Parallel coordinates plot obtained from the sweep of selected SERL(50) hyperparameters. Each configuration has been trained for $2 \cdot 10^6$ frames.

# 6 Mutation Comparison

The contribution of the safety-informed mutation operator has been specifically adapted for the current learning task so its relative contributions have been investigated in a comparative study. The present section presents and discusses the results of this study. Section 6.1 looks at the global effect the mutation operators have on learning. Then, Section 6.2 estimates the relative changes in the offspring policy safety and performance over one generation.

In both sections, the tail probability $p$ is calculated using the $t$-test and the null hypothesis is rejected for $p < 0.05$. For both analyses, the SERL(10) configuration has been used with the same hyperparameters as in Part I.

## 6.1. Effect on training curves

First, the effect of each mutation operator is recorded during training. One can consider the average reward over the population of actors to be an indication of performance robustness. Thus, the time histories of this average, recorded over the entirety of training, are to be compared.

The policy-parameter sensitivity $s$ is calculated, as defined in Part I, over the samples from the critical buffer of the respective policy. To stably compute it, the size of the individual's critical buffer has to be strictly larger than the mutation batch size. Depending on how 'unsafe', the initially randomised policies are, collecting enough critical samples might require many evaluations. Empirically, it has been observed that at the mark of $4 \cdot 10^5$ training frames, there are enough experiences stored by the critical buffer to stably compute the parameter sensitivity value.

Figure 6.1 shows the evolution of the population average during training as affected by three mutation operators with the same intensity $\sigma_{mut}$. By visually comparing the learning curves, one can see that both sensitivity-based mutations consistently outperform the more simple Gaussian operator. Moreover, the safety-informed mutation achieves better performance than both the proximal and the Gaussian mutation. Nevertheless, the proximal mutation still obtains higher returns with respect to the normal mutation. Thus, the safety-informed mutation contributes towards more stable learning which can be attributed to a lower rate of actors forgetting due to a disruptive change in their parameters.

It is worth taking into account that a more performance-robust mutation can ultimately have a positive effect on the maximum performance of the champion. Figure 6.2 shows the learning curves corresponding to the best-behaving actor. Empirically, one can argue that safety-informed exploration directs learning away from hard-to-learn-from experiences. In doing so, it develops policy functions with smoother loss landscapes which would have a stable learning process. Such policies would benefit from enhanced sample efficiency, converging towards better controllers than the ones that incorporate knowledge of the unsafe region in an unbiased manner.

Nevertheless, the information from the unsafe experiences is still learnt in the TD3 actor-critic structure from the gradient descent steps performed on transitions randomly sampled from the shared memory buffer.

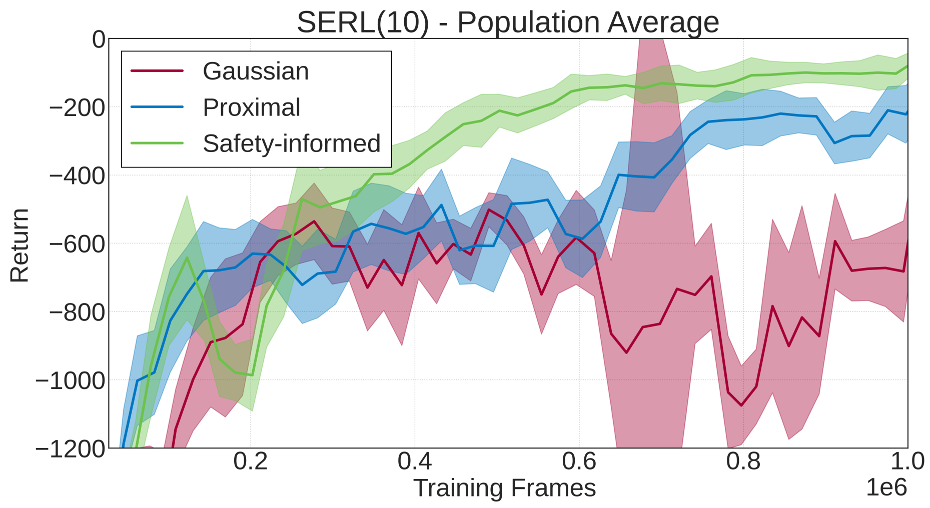


**Figure 6.1:** Learning curves of the average over the actor population trained with each of the three mutation operators. The reward statistics are calculated over $5$ evaluation runs at each training step.

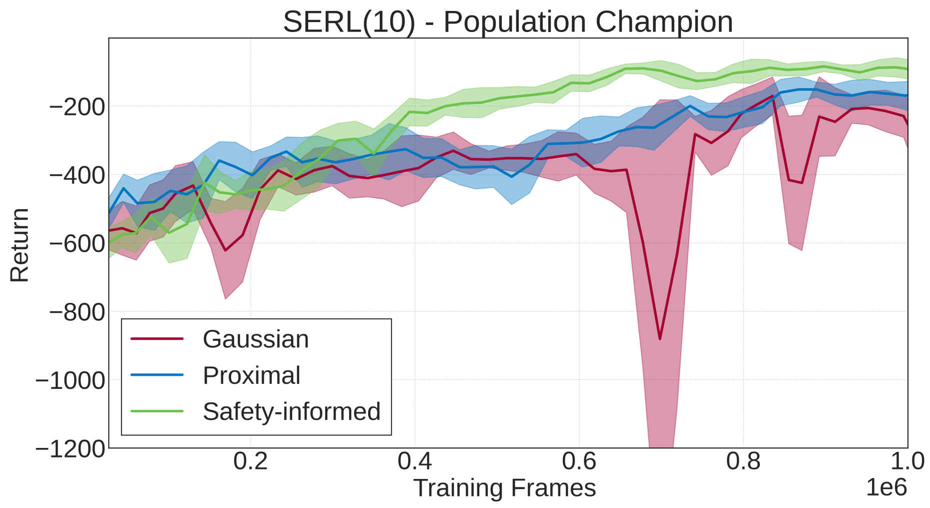


**Figure 6.2:** Champion learning curves for the three mutation operators. The reward statistics are calculated over $5$ evaluation runs at each training step.

## 6.2. Generation Statistical Analysis

Merely inspecting the learning curves does not provide enough insight with respect to the level at which the safety-informed mutation accomplishes its goals. Namely, the operator shall reduce the chance of catastrophic forgetting and pursuing unsafe behaviours.

The current analysis considers the effect of each mutation operator after one generation. Each actor within the population mutates with each operator, resulting in three offspring. The reference signals used to evaluate the parent and each offspring have been generated from the three seeds and are saved for re-use. After evaluating the offspring, using Equation 6.1 one calculates the relative change in episodic return and size of the critical buffer.

$$\Delta_{r/c} = \frac{offsping_{r/c} - parent_{r/c}}{|parent_{r/c}|} \tag{6.1}$$

The distribution of the relative changes in return is plotted in Figure 6.3 and the size of the critical buffer in Figure 6.4. The latter can also be understood as the relative change in the total number of unsafe transitions recorded by the actors before and after mutation.

Over one generation, the Gaussian mutation obtained a significantly larger performance gain than the proximal ($p = 6.3 \cdot 10^{-10} < 0.05$) and the safety-informed mutation ($p = 7.08 \cdot 10^{-6} < 0.05$). The safety-informed operator also has an average variation in performance that is better than the proximal operator ($p = 0.004 < 0.05$).

When comparing the minima of each box plot from Figure 6.3, the most extreme loss in reward is shown by the proximal operator. Meanwhile, the safety-informed mutation obtains the best worst-case performance. As catastrophic forgetting is associated with a significant and negative score, the current safety-informed operator reduces its chances compared to the Gaussian and proximal operators.

The Gaussian operator adds zero-mean symmetric parameter noise. Intuitively, it does not distinguish between transitions that can result in noisy or unsafe behaviours. The first box plot from Figure 6.4 confirms this intuition as the relative change number of unsafe transitions of the Gaussian mutation is symmetrically distributed, with the average change close to zero (at $0.0018$). The proximal mutation penalises sudden shifts in the policy but it does not contain any domain knowledge. Still, it achieves more conservative offsprings, with the critical buffer sizes lower on average than the Gaussian mutation ($p = 5.38 \cdot 10^{-8} < 0.05$).

Looking at the third box plot from Figure 6.4, injecting even rudimentary domain knowledge can improve the safety of the offspring policy. The safety-informed offspring has a critical buffer reduced by more than four times with respect to the Gaussian mutation and by $50\%$ when compared to the proximal one.

To conclude, the analysis shows that, over one generation update, the safety-informed mutation has statistically significant gains in terms of reducing unsafe transitions while limiting the chance of catastrophic forgetting.

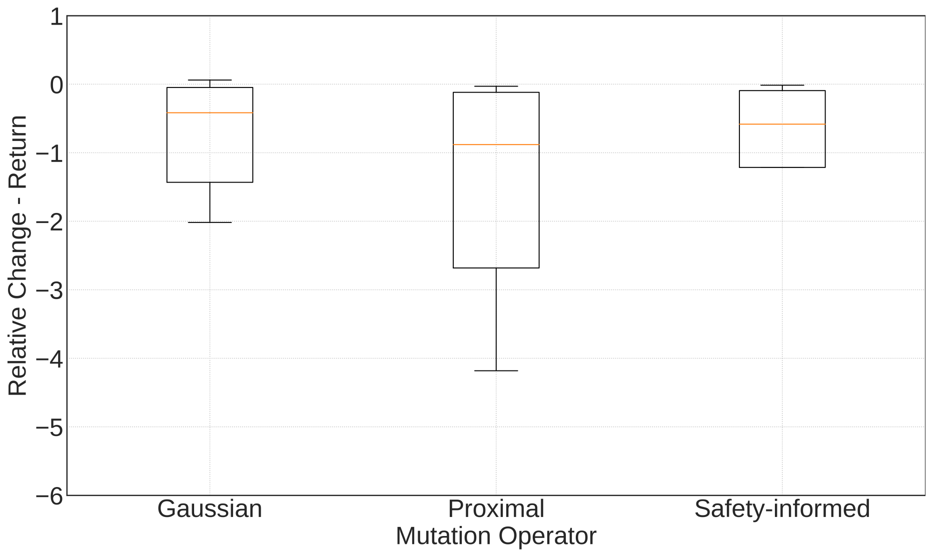


**Figure 6.3:** Relative change in episode return after mutating. Statistics are calculated for 10 actors over the same 30 trials. A negative score represents a loss in performance after the update.

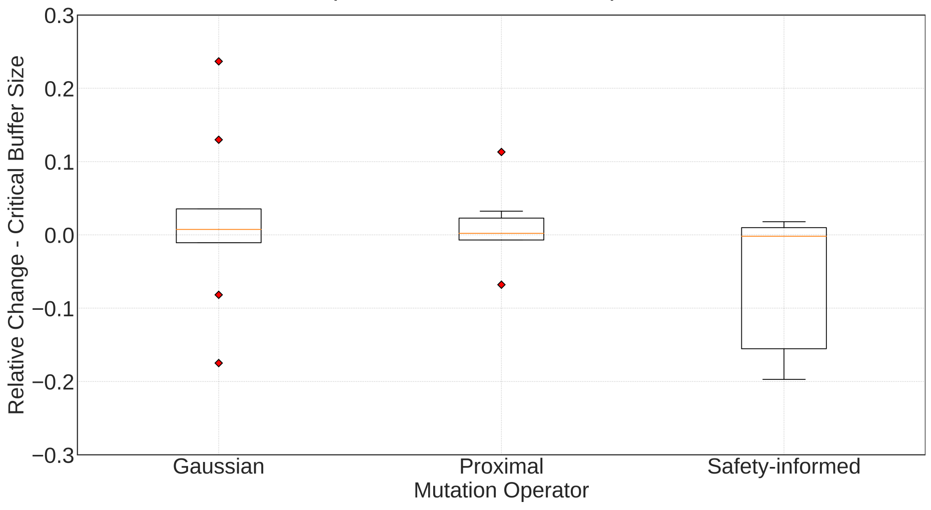


**Figure 6.4:** Relative change in the size of the critical buffer over episode obtained after mutating. Statistics are calculated for 10 actors over the same 30 trials. A positive score corresponds to a less safe offspring.

# 7 Additional Evaluation Responses

In Part I, a comparative analysis between TD3, SERL(10) and SERL(50) was presented in terms of their robustness against faults, unseen conditions and external disturbances. The present chapter lists the additional time responses obtained during this analysis which have not been presented before. To enable a straightforward visual inspection, the time histories plotted have been generated on the same reference signals. They serve as an extension of the results presented and discussed by Part I and thus aim to further document the difference between the control policies trained by SERL and TD3.

Table 7.1 shows the average performance values for the population average and the champion actor when tested on the fault-tolerance and robustness evaluation scenarios.

**Table 7.1:** Evaluation nMAE and standard deviation for the TD3 and SERL(50) population on the ten scenarios. The bold-faced values denote an nMAE significantly lower than the rest ($p < 0.05$).

| Identifier | Case | TD3 | Average SERL(50) | Champion SERL(50) |
|---|---|---|---|---|
| - | Nominal | $7.43 \pm 1.6\%$ | $6.55 \pm 1.5\%$ | $\mathbf{4.14 \pm 1.0\%}$ |
| **F1** | Iced Wings | $7.89 \pm 1.5\%$ | $7.59 \pm 2.4\%$ | $\mathbf{4.73 \pm 0.8\%}$ |
| **F2** | Shifted Centre of Gravity | $7.41 \pm 1.4\%$ | $7.87 \pm 2.0\%$ | $\mathbf{4.80 \pm 0.8\%}$ |
| **F3** | Saturated Aileron | $7.52 \pm 1.6\%$ | $6.55 \pm 1.5\%$ | $\mathbf{4.14 \pm 1.0\%}$ |
| **F4** | Saturated Elevator | $8.04 \pm 1.9\%$ | $7.07 \pm 1.4\%$ | $\mathbf{4.74 \pm 1.5\%}$ |
| **F5** | Broken Elevator | $8.63 \pm 1.9\%$ | $13.68 \pm 6.2\%$ | $8.08 \pm 1.3\%$ |
| **F6** | Jammed Actuator | $\mathbf{37.31 \pm 0.5\%}$ | $73.97 \pm 21.3\%$ | $39.58 \pm 0.9\%$ |
| **R1** | High Dynamic Pressure | $10.06 \pm 0.8\%$ | $8.201 \pm 3.3\%$ | $\mathbf{4.15 \pm 0.5\%}$ |
| **R2** | Low Dynamic Pressure | $11.53 \pm 2.2\%$ | $11.36 \pm 5.3\%$ | $\mathbf{6.68 \pm 1.6\%}$ |
| **R3** | Disturbance & Biased Sensor-noise | $7.13 \pm 1.4\%$ | $6.94 \pm 1.5\%$ | $\mathbf{4.74 \pm 1.2\%}$ |

## Nominal conditions

The agents are tested using the same trim condition as during training, namely $H = 2,000\ m$, $V_{tas} = 90\ m/s$. The system's responses, when controlled by the intelligent controllers, can be seen in Figure 7.1.

For the SERL population, the champion in nominal conditions corresponds to the one identified during training even though the error metric. While during training the champion is identified via tournament selection based on its undiscounted episodic return, in this case, the identification metric corresponds to the highest nMAE which does not account for low sideslip values.

## F1: Ice on wings

Ice developing on wings is a challenging fault as it evolves progressively which makes its detection more difficult. As conservatively modelled by the F1 fault case, it results in a down-sift of the lift curve and an increase in the skin friction drag coefficient. Here, the maximum angle of attack was reduced by $30\%$ and the drag coefficient increased by $0.06$. Figure 7.2 shows the time histories obtained during an evaluation episode by the SERL(50) champion (top plots) and the TD3 policy (bottom).

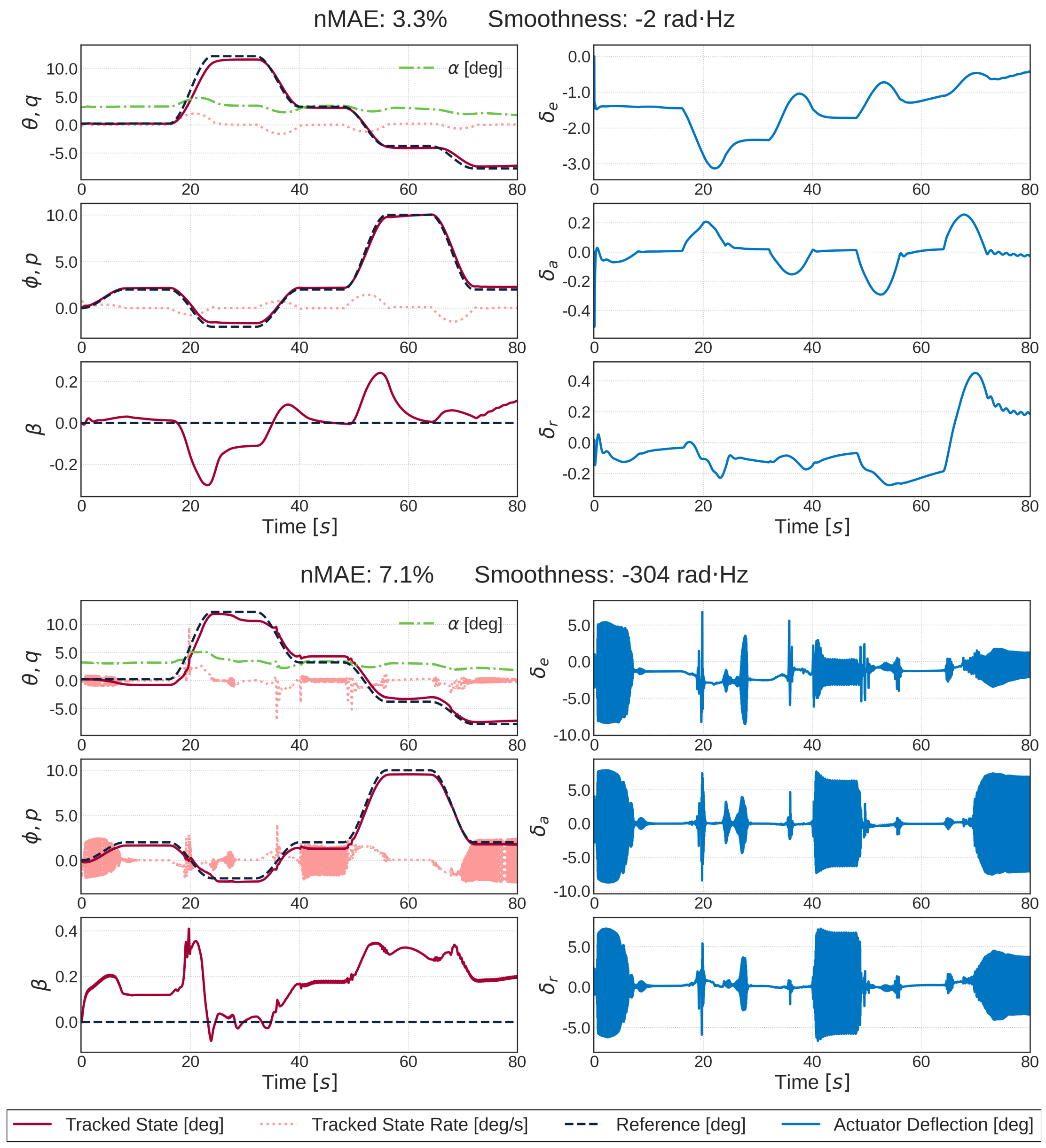


**Figure 7.1:** Evaluation time-traces of the champion (top) and TD3 (bottom) policy with their corresponding one-episode nMAE and $Sm$ metric for the **nominal** case.

The champion is different from the nominal case. Despite this, it is controlling the modified plant dynamics with a similar tracking error and it commands more negative elevator deflections. The TD3 actor shows, on average, the same change in behaviour but the high degree of action noise makes it less visible in the time trace of the elevator deflection.

## F2: Aft-shifted centre of gravity

Shifting the CG aft by $0.25\ m$ pushes the aircraft closer to its static stability margin. Such dynamics are still controllable via feedback control but the response can become unstable during pitch-up manoeuvres.

Looking at Figure 7.3, the SERL champion counteracts the longitudinal moment caused by the fault with a positive elevator defection. By doing so, the SERL champion offers more accurate pitch tracking with respect to the SAC actor trained by [18]. Meanwhile, the lateral responses remain comparable.

The fault is expected to preponderantly affect the longitudinal axis. Due to the non-linear coupling present in the modelled dynamics and, subsequently, in the trained intelligent controller, its effects are

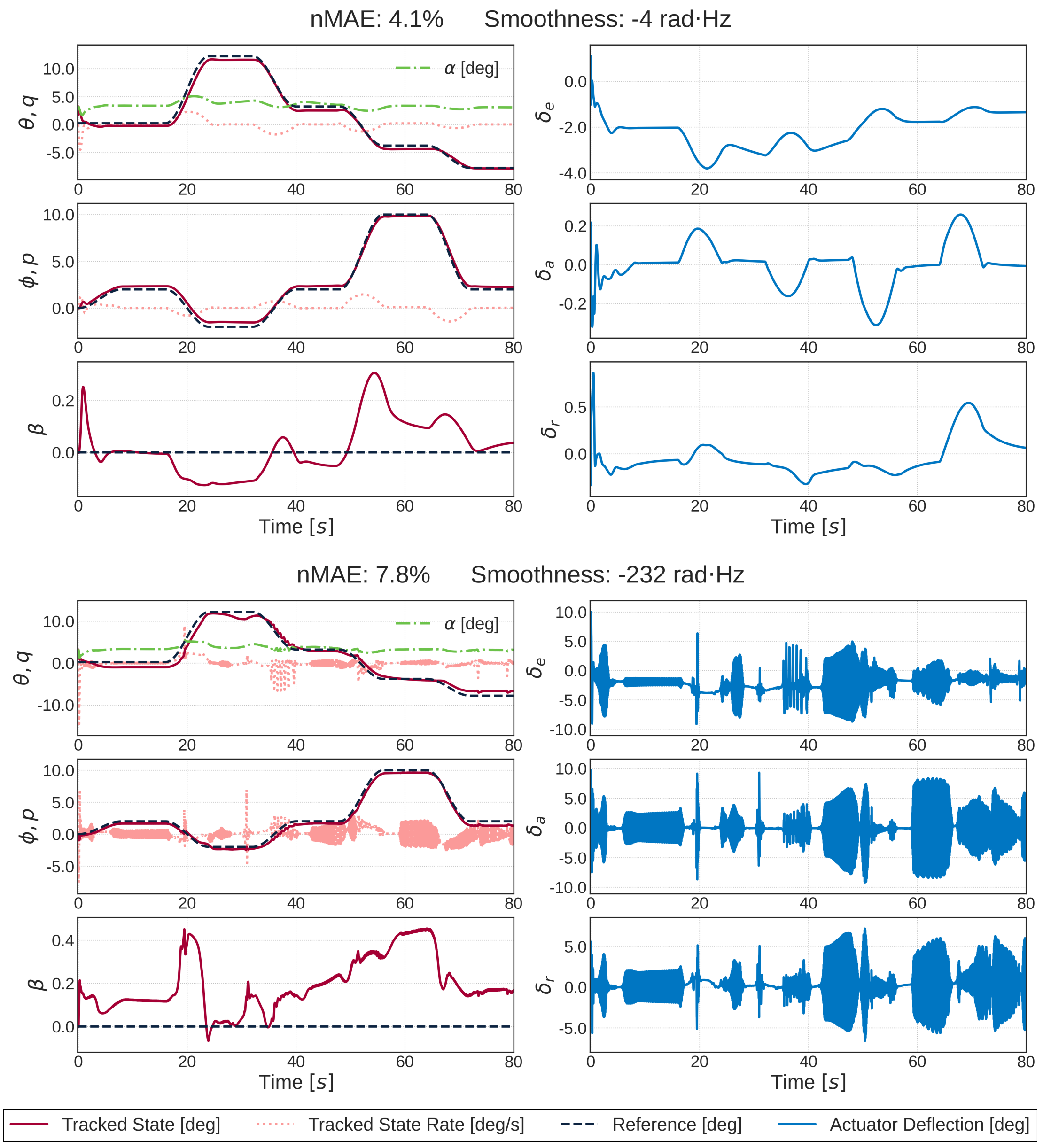


**Figure 7.2:** Evaluation time-traces of the champion (top) and TD3 (bottom) policy with their corresponding one-episode nMAE and $Sm$ metric for the **iced wings** case (F1).

visible on all three control channels. After pitch-up or down manoeuvres, the elevator command signal contains an oscillatory component, also traceable in the $\theta, q$ responses. This oscillation is then present in the roll and sideslip channels but it eventually damps out. Nevertheless, its magnitude and frequency are visibly lower than the oscillations present in the TD3 signals which suffer from the same action noise as in the other cases.

## F4: Saturated elevator

When faced with a low elevator saturation level, the controllers' tracking performance deteriorates. Figure 7.4 shows the responses of the population champion and the TD3 actor when tasked to control the aircraft with its elevator saturated at $\delta_e = \pm 2.5°$.

Comparing their responses, one can see that the commands computed by the SERL actor are more often within the saturation bounds compared to the TD3 actions. Therefore, the champion can on average better follow the pitch reference. As the trim elevator defection sits at $\delta_{e_{trim}} = -1.43°$, the symmetric

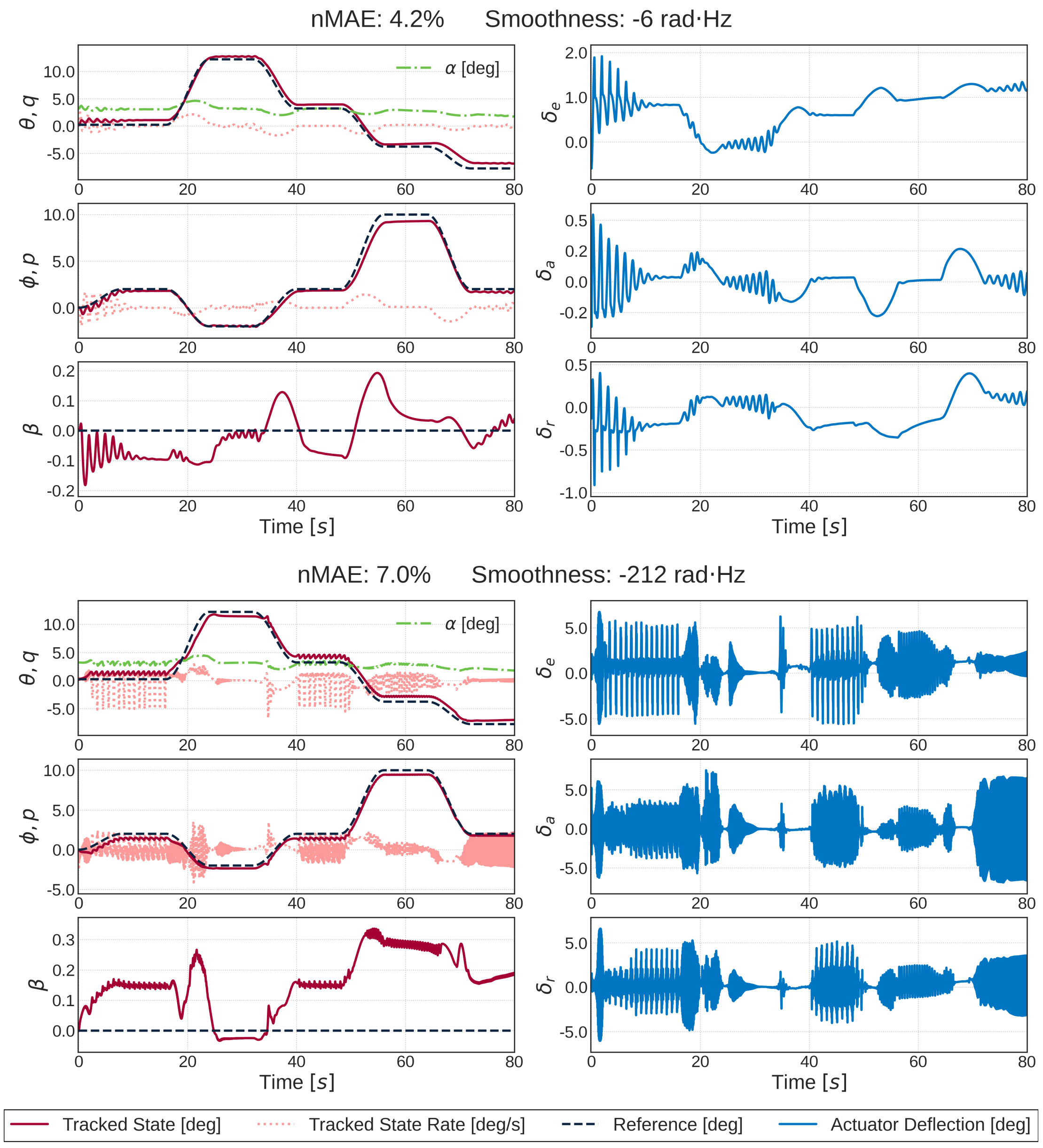


**Figure 7.3:** Evaluation time-traces of the champion (top) and TD3 (bottom) policy with their corresponding one-episode nMAE and $Sm$ metric for the **aft-shifted centre of gravity** case (F2).

saturation limits would affect more the negative commands, thus hindering steep pitch-up manoeuvres. This effect is directly visible for both SERL and TD3 starting from $t = 20\ s$.

Notably, the champion actor, in this case, is not one best-behaving actor in nominal conditions. The latter uses the elevator more aggressively, which becomes sub-optimal for these faulty conditions. This signals the relationship between the evolved diversity and the ability of the entire population to remain robust against faults and changes.

## F5: Broken Elevator

A partial loss of the horizontal surface can be caused by tail strikes during taking off. Such a fault triggers a sharp and sudden decrease in longitudinal control effectiveness. The current evaluation case models it by multiplying the elevator coefficients by a $0.3$ gain. Figure 7.5 shows the aircraft responses when controlled by the two agents.

Same a before, the TD3 actor behaves more aggressively across all actuating channels, with high-

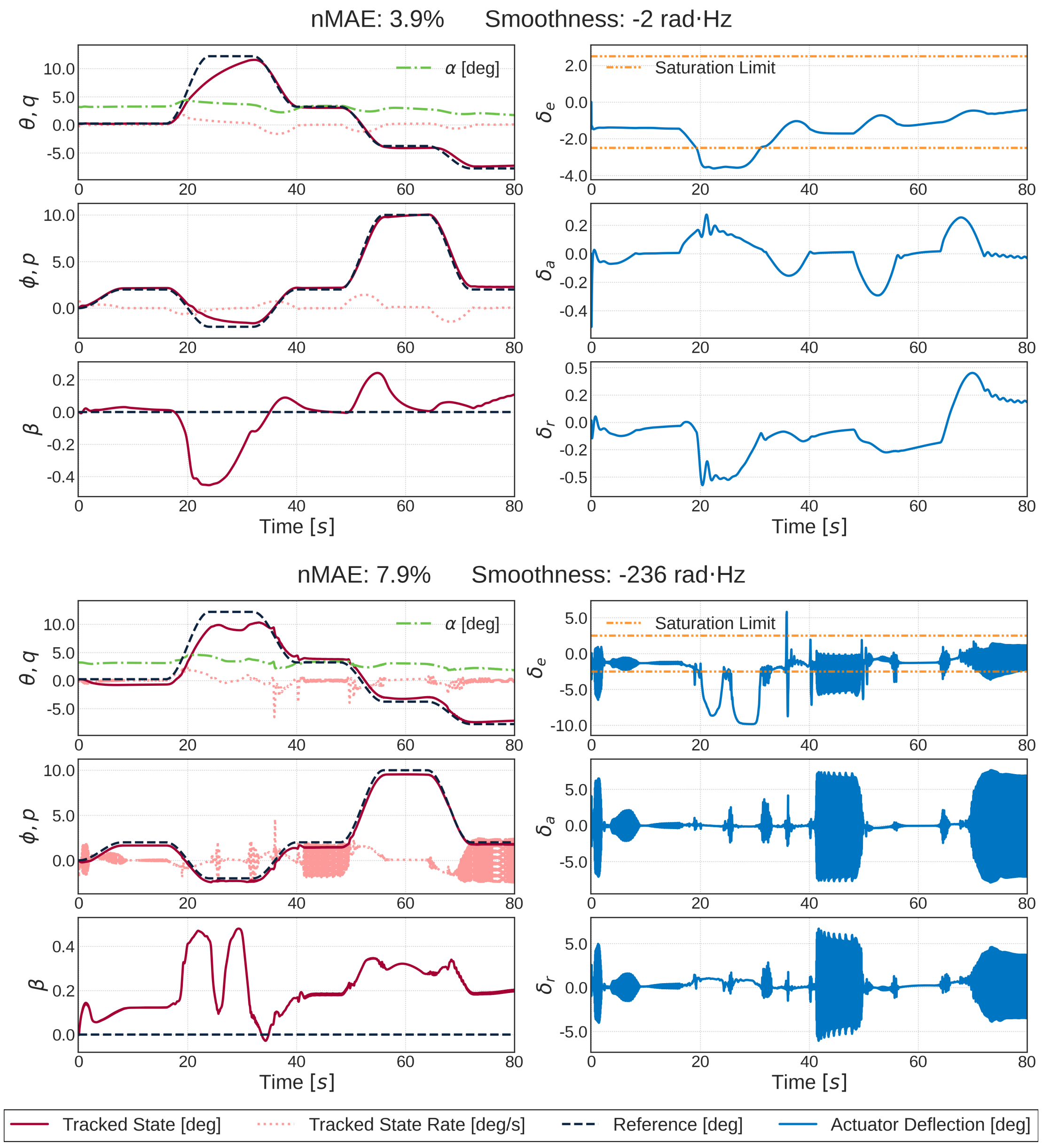


**Figure 7.4:** Evaluation time-traces of the champion (top) and TD3 (bottom) policy with their corresponding one-episode nMAE and $Sm$ metric for the **saturated elevator** case (F4).

frequency commands and large deflections. Whereas this has been considered an undesired phenomenon for deployment on real hardware, it can become beneficial when dealing with less effecting actuators. By comparing the plots from Figure 7.5, TD3 can better track the pitch reference in the presence of a broken elevator.

The SERL(50) champion is incapable of following a high-pitch reference even with a maximum elevator deflection of $-10°$ (reached at $t = 35\ s$) which significantly increases its nMAE. Furthermore, the non-linear coupling proves detrimental as, at that time stamp, both the roll and sideslip axis are negatively affected.

Despite this, SERL still maintains better sideslip tracking which results in a marginally lower nMAE than TD3.

## R1: Trim at high dynamic pressure

Previous works have proved robustness to the flight condition of a stochastic policy trained within a hierarchical SAC architecture and exposed to one trim condition [18, 108]. Despite this, their outer loop

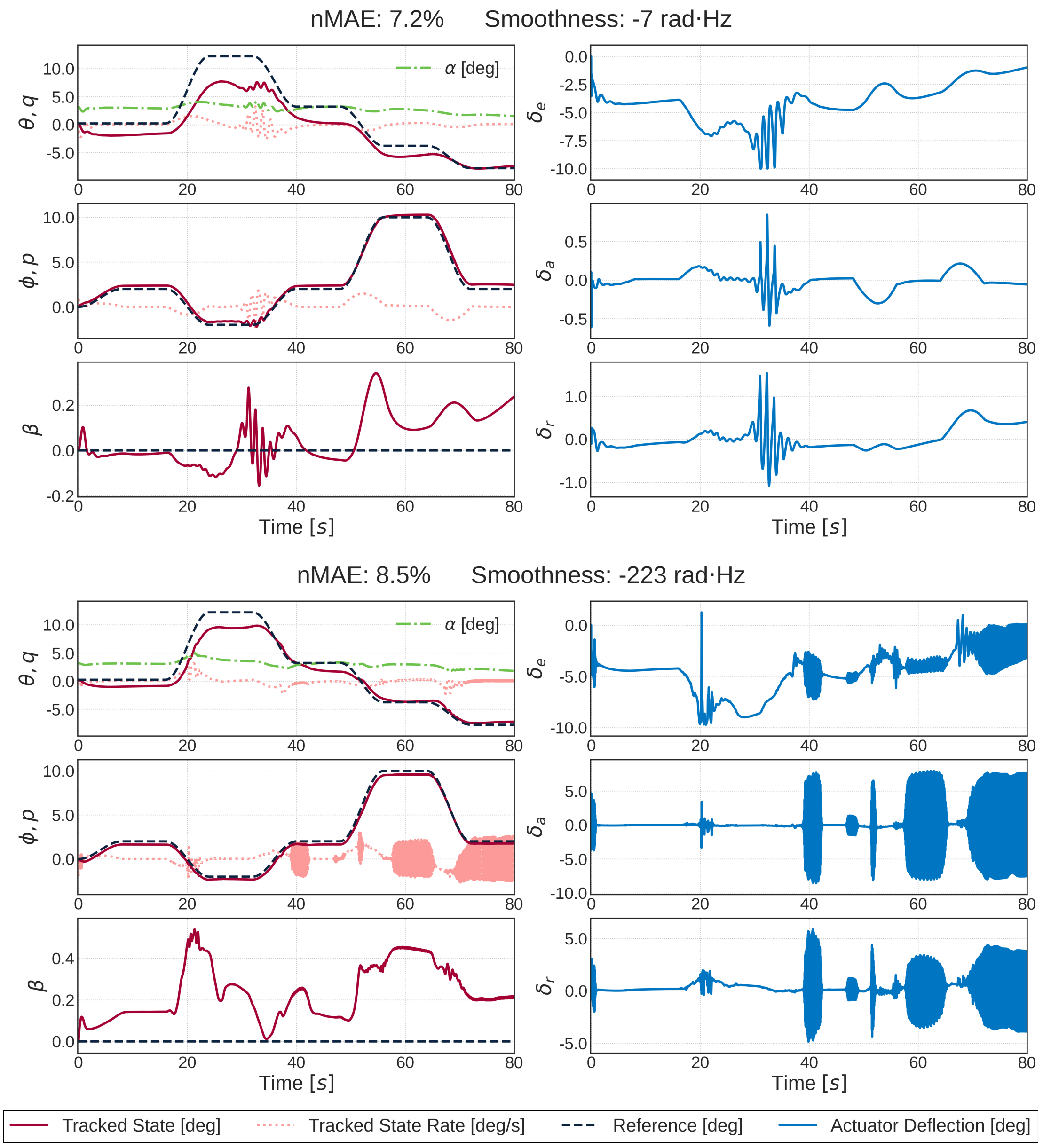


**Figure 7.5:** Evaluation time-traces of the champion (top) and TD3 (bottom) policy with their corresponding one-episode nMAE and $Sm$ metric for the **partially broken elevator** (F5).

controller observed the altitude which offers partial information with respect to the dynamic pressure. The plots from Figure 7.6 depict the time responses for the high dynamic pressure trim at $V_{tas} = 150\ m/s$ and $H = 20,000\ m$.

High dynamic pressure would increase the effectiveness of all three control surfaces. Thus, for the same manoeuvre, the required deflections are smaller. This proves beneficial for the SERL controllers, with the SERL(50) error deviation decreasing with respect to the nominal case. Moreover, the more agile plant contributes to less smooth control actions being commanded. Again, the TD3 policy is too aggressive, a highly sub-optimal trait when the controlled actuators become more effective.

## R2: Trim at low dynamic pressure

For this case, the simulated aircraft starts the evaluation episode at $H = 10,000\ m$, $V_{tas} = 90\ m/s$. In general, flying under low dynamic pressure conditions makes the responses sluggish. Since all three actuating channels become less effective, reacting to the change proves problematic for all controllers

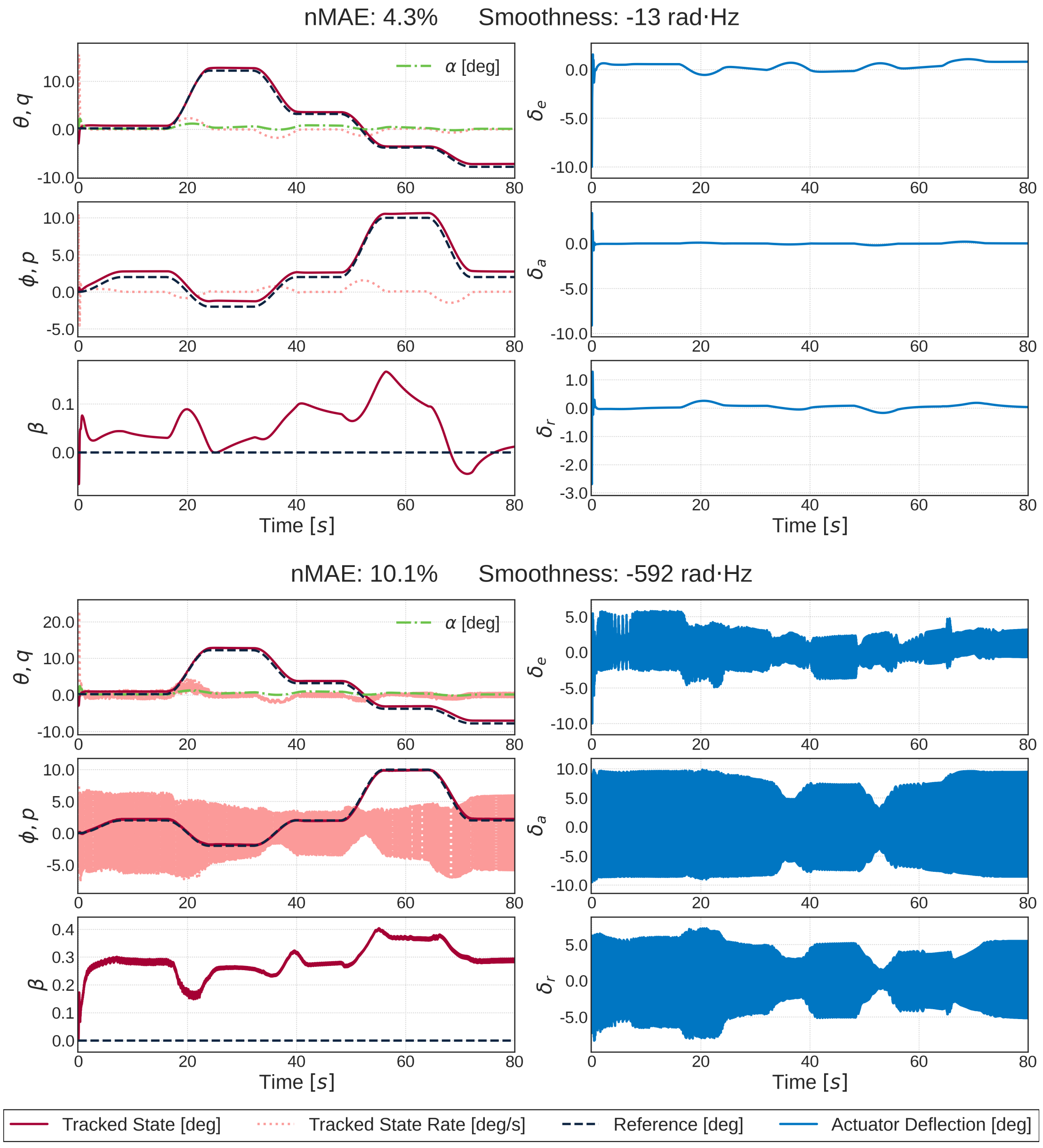


**Figure 7.6:** Evaluation time-traces of the champion (top) and TD3 (bottom) policy with their corresponding one-episode nMAE and $Sm$ metric for the **high dynamic pressure** trim (R1).

and translates into a higher average tracking error and standard deviation.

The plots of Figure 7.7 show the aircraft time responses. The RL-only agent remains more robust (when compared to the nominal case) in high altitude conditions because it benefits from the sluggish controllers. Despite this, it struggles to command the rudder, resulting in a relatively high sideslip error which drives its nMAE lower than the SERL(50) champion.

Looking at the beginning of the top plots of each set from Figure 7.7, both actors initially pitch down rapidly, by commanding positive elevator defections. This is done to align the aircraft pitch, initially high due to the low-pressure trim setting, with the reference value which starts at zero. The SERL champion only requires a quarter of the maximum deflection admissible whereas the TD3 actor maximises the actuator.

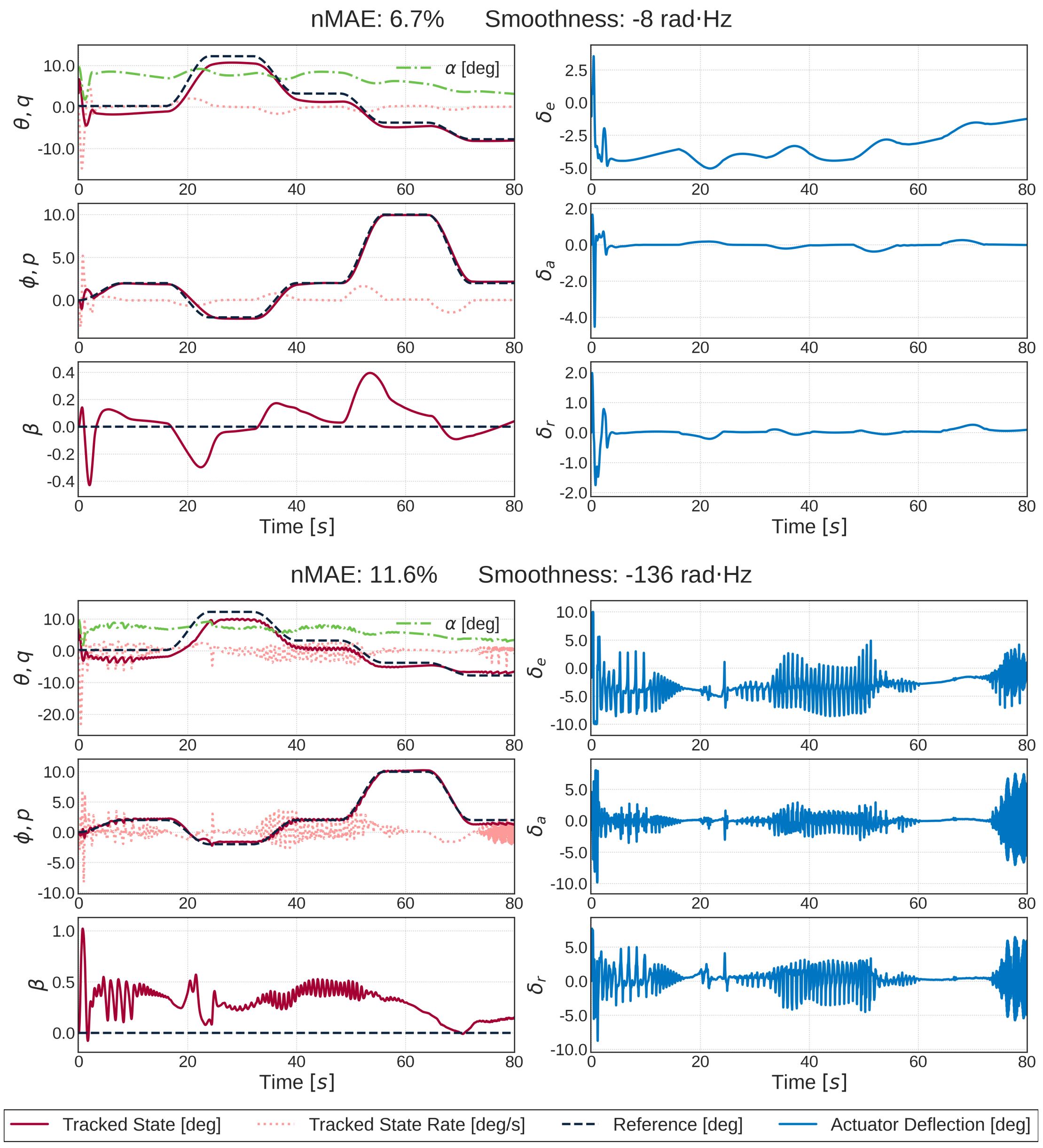


**Figure 7.7:** Evaluation time-traces of the champion (top) and TD3 (bottom) policy with their corresponding one-episode nMAE and $Sm$ metric for the **low dynamic pressure** trim (R2).

## R3: Atmospheric disturbance and sensor noise

Figure 7.8 shows the system response in the R3 case which considers an external disturbance according to MIL-F-8785C specification from [109] in the presence of realistic sensor noise, presented in Table 7.2 from [110]. The aircraft is subjected to a vertical wind gust of $15\ ft/s$ lasting from $t = 20\ s$ to $t = 23\ s$. As for this trim condition, an angle of attack increase is expected to be more problematic than a decrease, only an up-pointing gust has been considered.

Initially, the gust triggers a pitch-up motion and the SERL controller responds by instantaneously reducing the negative elevator deflection by $\Delta\delta_e \approx 1°$. By doing so, it rejects the disturbance, pointing the aircraft to return to the nominal pitch trajectory. Whereas the same behaviour is shown by TD3 (the top-left plot from the bottom set of Figure 7.8), the change in elevator deflection is less significant in comparison to the action noise. In this case, the SERL champion remains the same as for the nominal scenario, hinting that, in contrast to most of the faulty cases, genetic diversity is not necessary to reject the atmospheric disturbance.

Interestingly, this nominal trajectory (see Figure 7.1) shows a negative offset with respect to the reference pitch. Thus, even though the additional increase in pitch due to vertical gust would have brought the system closer to the reference trajectory, both controllers rejects it. A possible explanation stems from the angle of attack being part of the observed features. Both SERL and TD3 see their sudden increase as possibly 'unsafe', with the first being more prone to act against it due to the safety information encountered during evolution. Indeed, the SERL champion reduces the increase promptly and the TD3 actor accommodates it.

Lastly, adding biased noise does not significantly influence tracking performance. After $t = 23\ s$, both controllers follow the reference with comparable tracking error as in the nominal case. As expected, the action smoothness decreases but it does not change significantly as neither of the trained policies amplifies the observation noise.

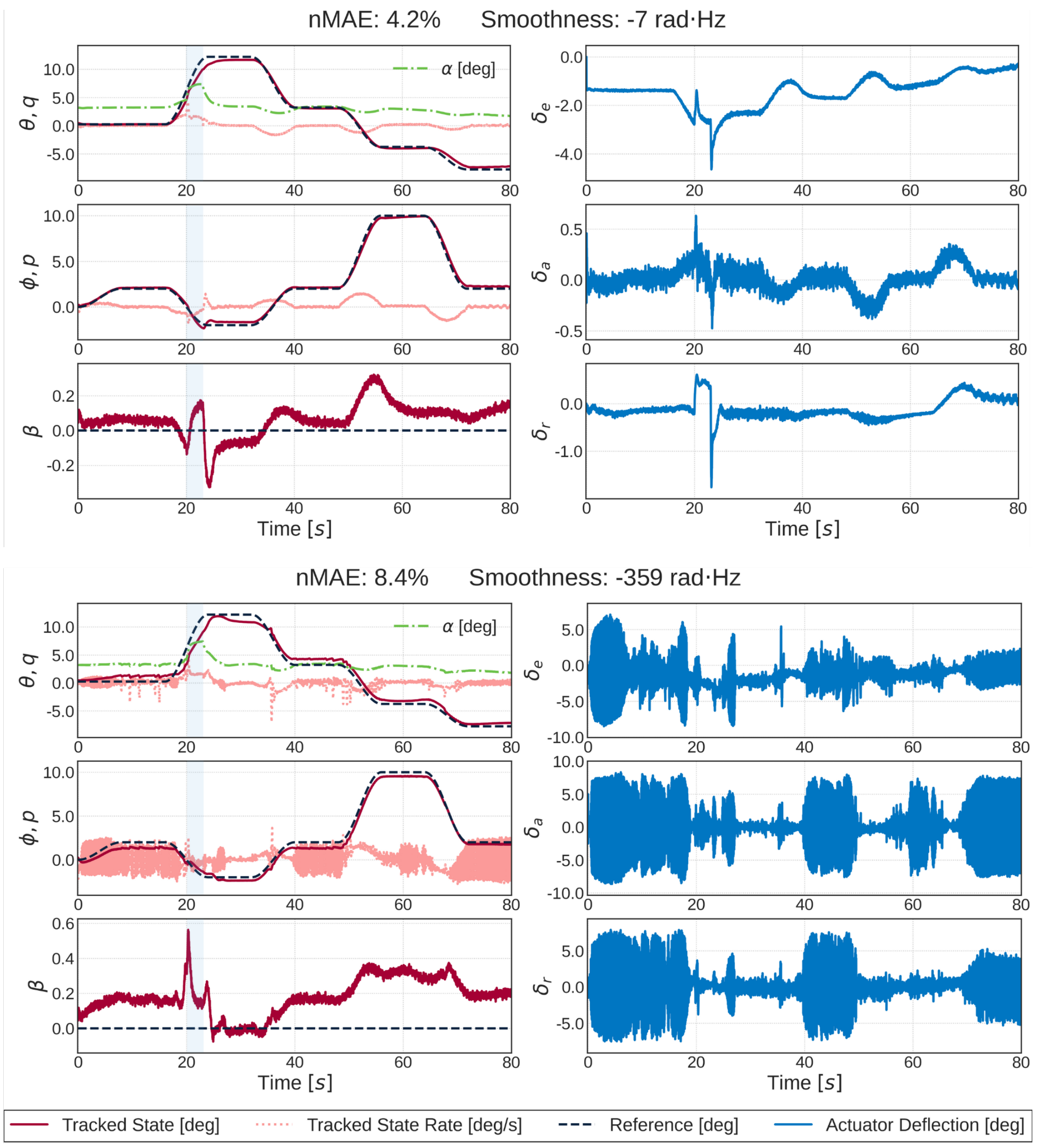


**Figure 7.8:** Evaluation time-traces of the champion (top) and TD3 (bottom) policy with their corresponding one-episode nMAE and $Sm$ metric when realistic sensor noise is applied and a $15\ ft/s$ vertical wind gust acts at $t \in [20\ s, 23\ s]$ (R3).

Overall, the performance of SERL-trained policies gracefully degrades when dealing with off-distribution

**Table 7.2:** Gaussian noise sensor models identified from flight tests on the PH-LAB. The relevant observations have been isolated from [110].

| Observed Signal | Unit | Noise Magnitude | Bias |
|---|---|---|---|
| $p, q, r$ | rad/s | $4.0 \times 10^{-7}$ | $3.0 \times 10^{-5}$ |
| $\theta, \phi$ | rad | $1.0 \times 10^{-9}$ | $4.0 \times 10^{-3}$ |
| $\alpha$ | rad | $4.0 \times 10^{-10}$ | - |

conditions, improving on the behaviours previously reported in the literature. The TD3 agent is less capable to reproduce its nominal performance once the conditions change. Nevertheless, the low dynamic pressure case remains the hardest for all controllers, pointing towards the need for such a trim setting to be included during training.

# 8 Adaptation via Online Selection

The hybrid framework generates an optimised population controller. Despite this, only one policy can control the plant at one time. Thus, for the framework to be useful in practical applications, a rule shall be designed to select which policy will control the system. The current chapter provides a proof of concept for such an adaptation rule based on online system identification.

To avoid later confusion, throughout this chapter the 'real' system will refer to the non-linear model within the DASMAT framework whereas the identified or modelled dynamics will correspond to the ones identified online.

First, Section 8.1 summarises the overview of online adaptation, followed by Section 8.2 which describes the mathematical background of online system identification based on recursive least squares. Then, Section 8.3 presents the proposed method of selection using the identified model. Lastly, Section 8.4 tests the online identification and selection method on the attitude reference tracking control task in the presence of a sudden fault.

## 8.1. Overview

One starts the adaptation process with the population of $N$ controllers trained offline according to the nominal training task. Initially, the nominal champion policy starts as the strongest candidate to act as the first behavioural policy. As the in-flight conditions are different from the ones encountered during training, the champion for one case is not guaranteed to be the same in other flight conditions or in the presence of unexpected faults. Hence, when deployed, the agent can increase its performance by switching the behavioural policy to adapt to the encountered conditions. The current work transforms the discrete decision-making into a receding horizon optimisation problem that requires online-learnt information about the new conditions and dynamics.

A sample-efficient way to perform online model identification uses an incremental model estimated via Recursive Least Squares (RLS). The incremental model provides a future estimate of the agent state to be used for trajectory prediction over the receding time horizon. Estimating the trajectory of each actor over the horizon and summing its reward can then inform the agent with respect to the best-behaving individual. This locally-relevant new champion switches with the incumbent (if required), becoming the new behavioural policy. Figure 8.1 presents, in high-level terms, the steps followed by the online adaption algorithm.

For this approach, only the model is learnt online while the behavioural policy is merely selected among the offline-trained controllers. The next section will further detail the model identification procedure.

## 8.2. Recursive Least Squares

While the aircraft system is continuous, the measured data are discrete samples. Given the high and constant sampling rate, the nonlinear continuous dynamics can be distressed according to Equation 8.1:

$$\begin{aligned} \mathbf{x}_{t+1} &= f\left(\mathbf{x}_t, \mathbf{a}_t\right) \\ \mathbf{y}_t &= h\left(\mathbf{x}_t\right) \end{aligned} \tag{8.1}$$

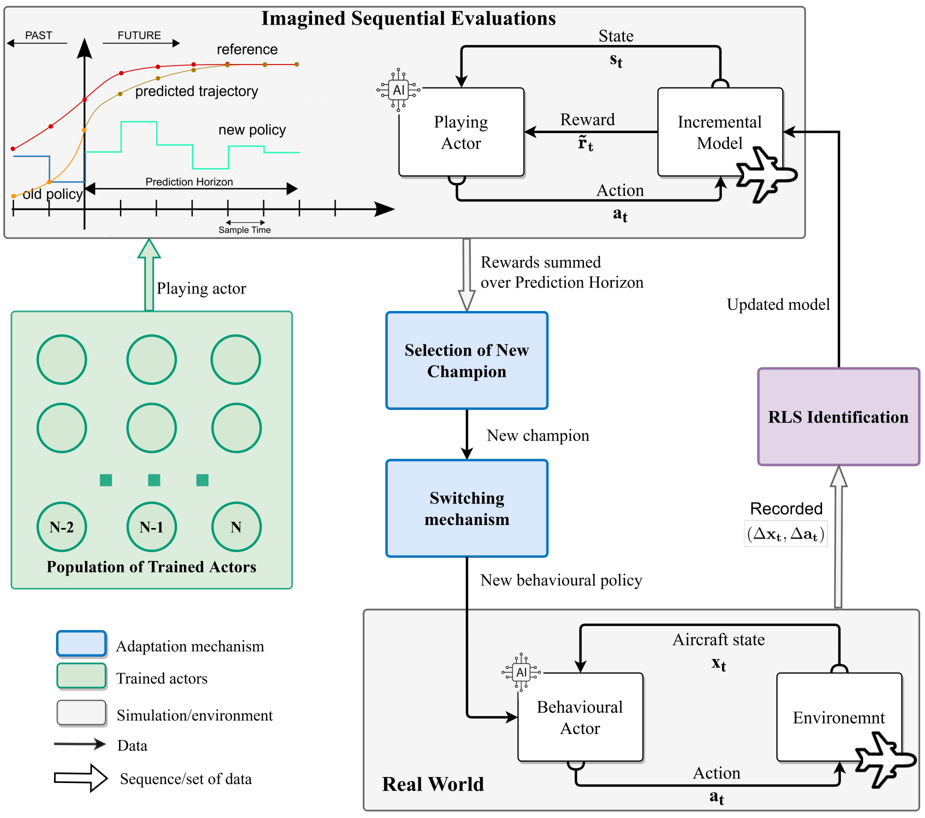


**Figure 8.1:** High-level overview of the online adaption mechanism. The top simulation block corresponds to the identified dynamics, whereas the bottom one symbolises the interaction between the behavioural policy and the real world.

For modelling the aircraft system, perfect state observability is assumed which translates into $h()$ becoming the identity function. So the output vector becomes identical with a sub-set of the DASMAT state (described by the Methodology section of the Part I):

$$\mathbf{y} \equiv \mathbf{x} = [p, q, r, V_{tas}, \alpha, \beta, \theta, \phi, \psi, H]^\top \in \mathbb{R}^{10}$$

The incremental model described by Zhou et. al. [111] uses a first-order Taylor series expansion to express the change in the system state based on the incremental control action. It was previously used in works on online adaptive flight control [6, 108]. Equation 8.2 depicts its state-update rule:

$$\Delta\hat{\mathbf{x}}_{t+1} = F_{t-1}\Delta\mathbf{x}_t + G_{t-1}\Delta\mathbf{a}_t \tag{8.2}$$

Where $F_{t-1} \in \mathbb{R}^{n\times n}$ is the system matrix and $G_{t-1} \in \mathbb{R}^{n\times m}$ the control effectiveness matrix. These time-varying matrices are updated by the RLS estimator. Since the control input $\mathbf{a} \in \mathbb{R}^m$ corresponds to control surface deflection and $\mathbf{x}$ to the reduced aircraft state, for the current work $n = 10$ and $m = 3$.

The RLS identification procedure collects the previously measured $M$ samples of the state and input increments in the measurement matrix $X_t$ from Equation 8.3 and the dynamics terms in the parameter matrix $\Theta_t$ from Equation 8.4. For the current implementation, the number of samples was limited to the previous step only, i.e., $M = 1$.

$$X_t = \begin{bmatrix} \Delta\mathbf{x}_t \\ \Delta\mathbf{a}_t \end{bmatrix} \in \mathbb{R}^{(n+m)\times M} \tag{8.3}$$

$$\Theta_{t-1} = \begin{bmatrix} F_{t-1}^\top \\ G_{t-1}^\top \end{bmatrix} \in \mathbb{R}^{(n+m)\times n} \tag{8.4}$$

Then, the recursive update rule is given by Equation 8.5, with $\kappa \in [0, 1]$ the forgetting factor, and it uses knowledge on the covariance matrix $P_{t-1}$. The symmetric $P_t$ matrix is a measure of the parameter-estimate covariance and is updated according to Equation 8.6 using the measurement innovation defined by Equation 8.7.

$$\Theta_t = \Theta_{t-1} + \frac{P_{t-1}X_t}{\kappa + X_t^\top P_{t-1}X_t}\boldsymbol{\epsilon}_t \tag{8.5}$$

$$P_t = \frac{1}{\kappa}\left[P_{t-1} - \frac{P_{t-1}X_tX_t^\top P_{t-1}}{\kappa + X_t^\top P_{t-1}X_t}\right] \tag{8.6}$$

Lastly, the prediction innovation from Equation 8.7 is the difference between the state predicted by the model and the measured one.

$$\boldsymbol{\epsilon}_t := \Delta\mathbf{x}_{t+1}^\top - \Delta\hat{\mathbf{x}}_{t+1}^\top = \Delta\mathbf{x}_{t+1}^\top - X_t^\top\Theta_{t-1} \tag{8.7}$$

Before the RLS starts to identify the dynamics, measurement and parameter matrices are initialised with zeros. The covariance starts with a scaled identity matrix since the method assumes no prior knowledge of the parameter values.

During online adaptation, the recursive updates are based on the experience generated by the behavioural policy. The next section will detail how the behavioural policy is changed using the newly learnt knowledge. Also, the implementation of the RLS identification was verified according to the procedure described in Chapter 9.

## 8.3. Online Selection

Let one assume a population of $N$ pre-trained actors which, on average, can control the system within a given performance requirement. Following previous chapters, this section takes the SERL(50) population which obtained an average nMAE of $6.55 \pm 1.4$% on the nominal tasks.

Using the RLS-identified model, the policies are played in the agent's 'imagination'. The RLS model updates continuously throughout the entire episode based on the measurements of the real dynamics. Every $t_{play} = 2\ s$, the current version of the identified model is used to predict the trajectory of each actor over a time horizon of $t_h = 2\ s$. Assuming unlimited computational power, this is assumed to happen on a separate thread from the computations of the behavioural policy which are thus not affected.

The reward summed over the prediction horizon becomes the actor's return. The individual that has the highest predicted return is then considered the predicted champion. This new champion replaces the behavioural policy if and only if its return is higher than the incumbent's by a margin of $50$%.

Once admitted for switching, two methods have been proposed to update the behavioural policy according to the new champion:

1. **Hard switch**: the champion parameters are copied in place of the parameters of the behavioural policy NN. The copy is made at the instant of the switching time.
2. **Soft switch**: the new champion (c) becomes a target for the behavioural actor (b). The latter's parameters are progressively updated using the Polyak moving average from Equation 8.8 [75].

$$\theta_b \leftarrow (1-\rho)\cdot\theta_b + \rho\cdot\theta_c \tag{8.8}$$

With $\theta_b$ denoting the parameters of the behavioural policy and $\theta_c$ the parameters of the newly selected champion actor. The Polyak coefficient $\rho$ has a value of $0.05$.

## 8.4. Adaptation to Faults

To test the fault-tolerance, a fault was inflected. As adaption has to be performed online, the scenario considers a sudden aft-wards longitudinal shift in the aircraft CG by $0.25\ m$. Since the shift happens at $t = 20\ s$, a longer evaluation episode of $T = 100\ s$ has been considered. Figure 8.2 shows the results of the two update methods alongside the case when there no online selection takes place. In the latter scenario (i.e., top pair of plots), the nominal champion remains the behavioural policy for the entity of the episode. The plots skip the lateral states as they are not directly affected by the longitudinal CG shift and, despite the observed non-linear coupling, they do not offer any further relevant insights about the behaviour of the adaptation method.

Both methods of online adaptation track the reference signals better than when the agent does not adapt. Their most notable contribution appears in the second half of the episode. After $t = 50\,s$, a relatively steep pitch-up manoeuvre (marked by a higher pitch rate) combined with the reduced static stability margin (caused by the shifted CG) triggers an undamped oscillatory behaviour. Whereas the nominal champion policy is affected by it, both adaptation methods reduce this oscillation at different degrees.

Overall, the soft switch method achieves the best performance, lowering the nMAE by more than $20\%$ from the case when the nominal champion remains the behavioural policy. The soft switch also improves the control policy smoothness, acting like a discrete low-pass filter in the policy parameter space which translates to more gradual transitions in the action space.

Despite this, the middle and bottom plots show multiple erroneous switches, for example at $t \approx 83\ s$ (middle plot) and $t \approx 50\ s$ (bottom plot). When the aircraft starts a new manoeuvre, the chance of exciting previously unencountered dynamics raises which inevitably increases the parameter covariance. This epistemic parameter uncertainty in the identified model transfers to the simulated trajectories and can result in an overestimation of the actors' reward, causing undesired switches. Whereas the short $t_h$ and the $50\%$ margin for switching specifically aim to reduce the frequency of unwanted changes, they can still occur.

This is considered an inherent limitation of the model identification method. Although sample efficient, the incremental model identified by the RLS retains accuracy in the neighbourhood of the encountered states. Thus, similar to the online RL methods, sufficient exploration of the state space enables better performance. According to [6], RLS depends less on global exploration but it benefits from the condition of persistent excitation. As no identification manoeuvres, such as the 3211, are specifically incorporated in the experiment, the locally identified dynamics are accurate for a short amount of time.

Last but not least, the above tests show that the adaptation tool can correct its mistake by switching back to another policy. In the pair of plots from Figure 8.2, the agent corrects for an erroneous selection made at $t \approx 83\ s$, hard-switching to another policy at the next play-time. As the soft switch gradually changes the behavioural policy, it dampens the impact caused by undesirable policy shifts due to reward overestimation.

## 8.5. Conclusion and Future Extensions

The current chapter presented the design and testing of a proof-of-concept for online adaptation. The tool incorporated Recursive Least Squares for fast online dynamics identification. The continuously-updating RLS model was used for each $t_{play}$ to play each actor over a receding prediction horizon of $t_h$. By summing the predicted reward over the horizon, the policy with the highest expected return was chosen as the new champion.

The chapter also presented two methods to switch the new champion with the incumbent behavioural policy - the soft and hard switches. Both methods obtained better nMAE and enhanced smoothness than the nominal champion when controlling a faulty aircraft. The soft switch method used a Polyak average to gradually shift the behavioural policy towards the new champion. By doing so, it showed the best tracking performance and the smoothest control signal. Thus, the test empirically proved the benefit of model-based online adaptation via selection over no adaptation mechanism in a fault-tolerance scenario.

The above results are impactful since they contribute to the limited body of knowledge concerning online population-based adaptation applied to flight control. Previous works either combined an offline trained policy with online updating [108], or developed dynamics-aware population-based searches on walking robots which benefit from relaxed safety constraints and can be easily rested in case the behavioural policy is incapable of fulfilling the task [33]

Following the above discussion, the present approach of adaptation via online selection could be improved in future work by following either one of these two recommendations:

1. **Uncertainty quantification**: The RLS indication can be replaced with dynamics models that can estimate the aleatoric and epistemic uncertainty (i.e. prediction uncertainty due to a lack of samples)

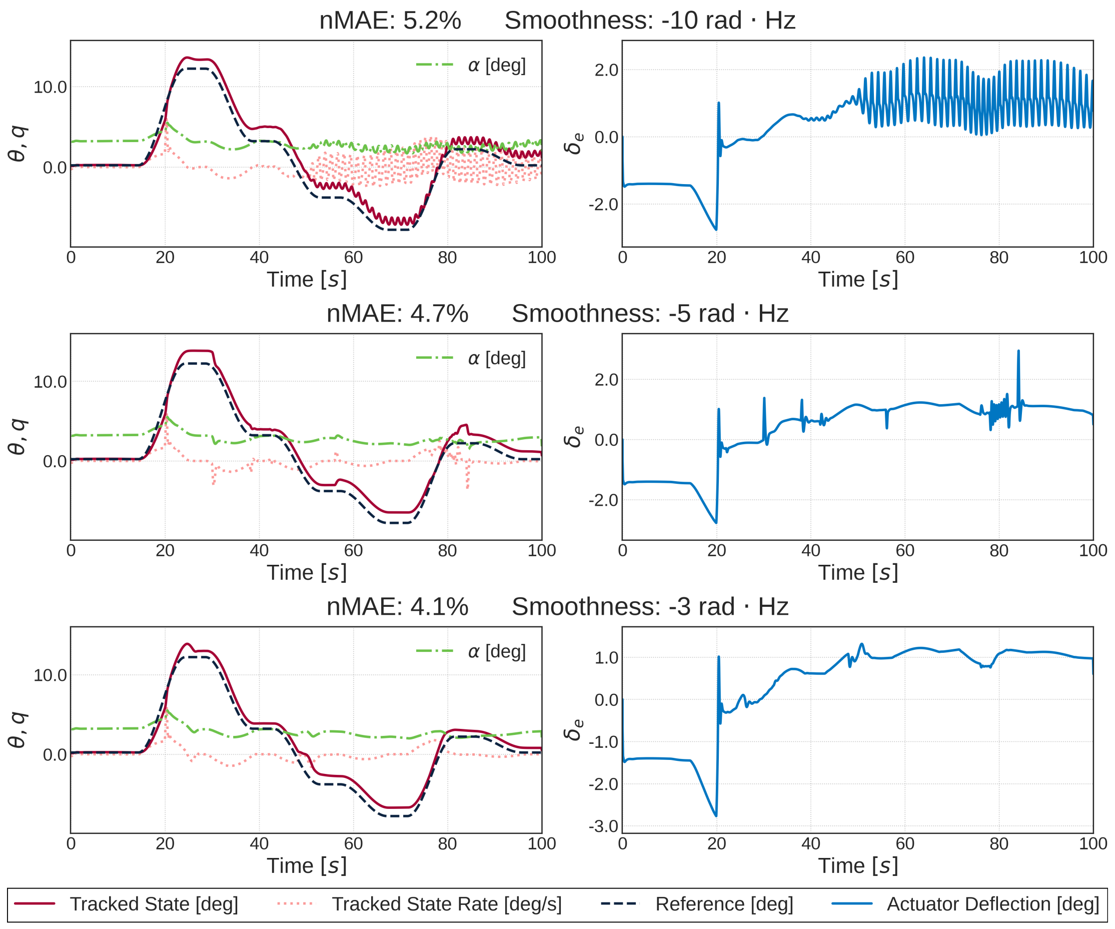


**Figure 8.2:** Evaluation time-traces of the pre-selected champion of the nominal case (top) and the two online switching schemes: hard (middle) and soft (bottom) with $\rho = 0.05$. At the time $t = 20\ s$, the aircraft's centre of gravity suddenly moves aft by $0.25\ m$.

over the time horizon. Possible candidates are Gaussian Process regression, pre-trained distributional critic(s) or ensembles of regression models. By doing so, the mechanism for online adaptation considers uncertainty and can select the policy which returns the best worst-case scenario. Despite this, where RLS can reasonably learn locally accurate dynamics, the sample efficiency of a more complex uncertainty-aware model is expected to deteriorate.

2. **Safety-constraints on offline policies**: The offline population can be tested against a set of safety constraints. By doing so, one can identify and remove the potentially unsafe ones. These schemes are not feasible for standard single-actor methods such as TD3 or SAC. Assuming the switching method will not violate the safety bounds, the offline-trained and online-selected behavioural policy could benefit from safety guarantees.

3. **Monte Carlo Tree Search**: At selection time, the agent has to take a discrete action within the finite space of policies which can also be assumed to follow a (PO)MDP. After identifying an uncertainty-aware prediction model with knowledge, one can exploit efficient probabilistic planners such as Monte Carlo Tree Search (MCTS) [33]. They have already been successfully used to solve POMDPs with stochastic transition functions [112]. This can lead to faster decision making in-between policies which becomes relevant when a large repertoire of offline-learnt policies is available.

# 9 Verification and Validation

Both the methodology and software tools used to generate the results are verified. This consisted of an iterative process of unit and module testing, performed at multiple levels of abstraction.

The developed source code is written in Python, with the PyTorch, Numpy and Scipy packages used within the training and testing scripts. Also, the Weight & Biases API has been adopted to log the training statistics and perform hyperparameter sweeps. Apart from that, the DASMAT framework is available as a Simulink model initialised by a MATLAB script.

For unit and module testing, the framework is split into three modules: the DASMAT model, the PDERL framework and the TD3 algorithm. The next three sections detail the unit testing procedure performed to verify each part. Apart from them, the online model identification presented by Chapter 8 is verified in Section 9.4

## 9.1. PDERL Implementation

Chapter 4 from Part II presents the implementation of the PDERL algorithm. Its goal is to prove the possibility of using PDERL to train policies for a continuous control task and remain tolerant against faults. For this purpose, the LunarLander environment served as a simplified problem to train a PDERL population and evaluate the intelligent controller.

Following directly from the conclusions made within the above-mentioned chapter, both the PDERL implementation and its ability to optimise a population of controllers are considered verified.

## 9.2. Learning Environment

As this module is wrapped inside the learning environment, the tests are performed pursuing a 'black box' approach. With this in mind, two tests have been completed.

First, the elevator channel is subjected to a positive step input while the aileron and rudder deflection is kept at zero. The simulated time responses are visible in Figure 9.1. As expected, the aircraft initially remain at the trim pitch but from $t = 5\ s$, the positive elevator input $\delta_e = 0.5°$ causes a pitch-down motion which subsequently triggers a slowly damped low-frequency oscillation, which corresponds to the phugoid eigenmode.

Second, a set of three PID controllers have been tuned to track the pitch, roll and sideslip references in nominal conditions (see the methodology section from Part I). Figure 9.2 depicts the time responses of the tracked aircraft states.

The PID controllers can track both the pitch and roll reference with little overshoot. The constant $\beta_r = 0$ reference is harder to follow due to present the asymmetric coupling. Despite this, the DASMAT-based environment reacts to control actions in a smooth manner which follows the standard principles of flight dynamics.

## 9.3. TD3 Learning Algorithm

The current TD3 implementation followed the algorithm developed by [13] and made publicly available on the page: `https://github.com/sfujim/TD3`.

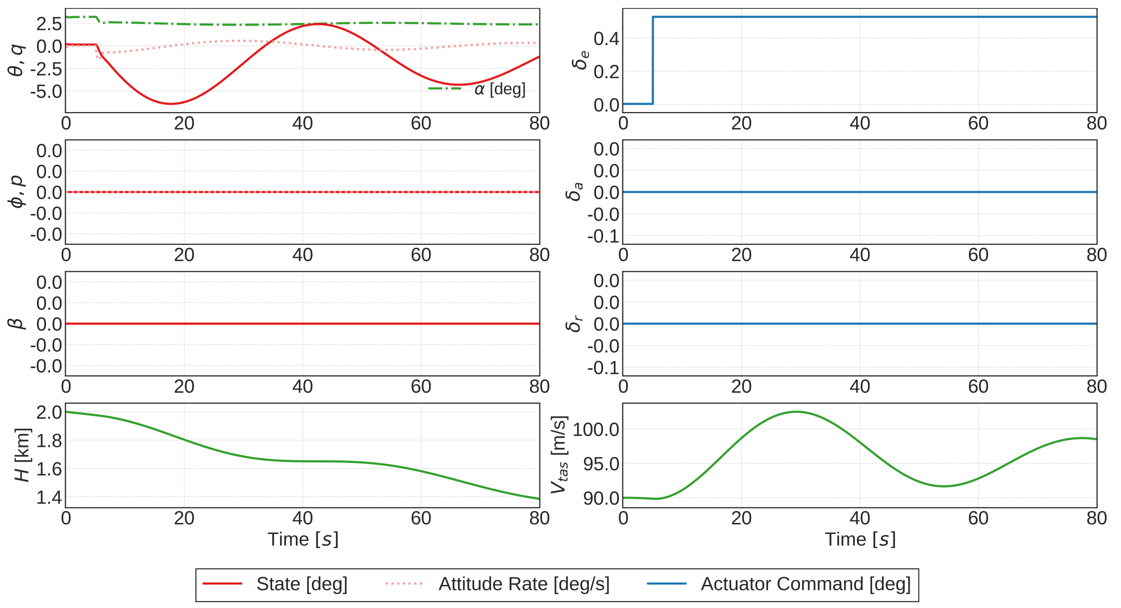


**Figure 9.1:** DASMAT response to a step signal fed through the elevator channel.

To verify the implementation, the agent is trained on a simplified control task using the already-verified non-linear aircraft model within the DASMAT framework. Whereas a similar methodology to the one described in Part I has been followed, this test considers learning a pitch reference tracking task. Figure 9.3a shows its corresponding control diagram. The agent shall learn to compute $\delta_e$ commands based on the pitch error $\Delta\theta = \theta - \theta_r$, and the longitudinal states $[q, \alpha, V_{tas}]^\top$ observed from the aircraft state vector. Previous actions or states are not considered, following the MDP assumption stated in Part I.

The reward function is scaled to account for the missing roll and sideslip channels and the reference corresponded to randomly generated sequences of cosine-smoothed steps in pitch. The progress statistics are subsequently logged. Figure 9.3b shows the average episodic return plotted against the number of training frames encountered.

After $400,000$ learning frames, TD3 has converged, tracking the pitch reference with an episodic return of $R = -30 \pm 6.6$. Thus, the resulting tracking performance is comparable with the performance obtained by a tuned PID controller and with one-third (due to pitch-only reward scaling) of the rewards obtained by inner-loop SAC agents trained for attitude control in [18, 108, 113].

Therefore, one can conclude that the results of this test verify the implementation of both the TD3 agent and its training algorithm.

## 9.4. Online Identification

The online adaption method introduced by Chapter 8 includes a model identification step via RLS, which, as part of the system identification process, has to be verified.

The current module test considers the same task presented in Chapter 8, namely aircraft control with a sudden aft-shift of the CG. Figure 9.4 shows the normalised one-step ahead prediction innovations. For this testing scenario, the behavioural policy corresponds to the pre-selected nominal champion policy.

The sudden backwards shift in the CG takes place at time $t = 20\ s$. The fault is visible in Figure 9.4 as a spike in each of the nine innovations. At the time of the fault, the pitch rate estimation error is the highest among all ten tracked states as the aircraft has already been performing a relatively steep nose-up manoeuvre when the CG shift causes an additional positive pitch moment.

By visual inspection, the RLS prediction errors are small (i.e., comparable to machine precision) and decreasing. Despite this, when new dynamics are excited some states become harder to predict. The pitch-up manoeuvres cause the velocity to decrease faster than the predicted value. Similarly, the angle of attack varies unpredictably.

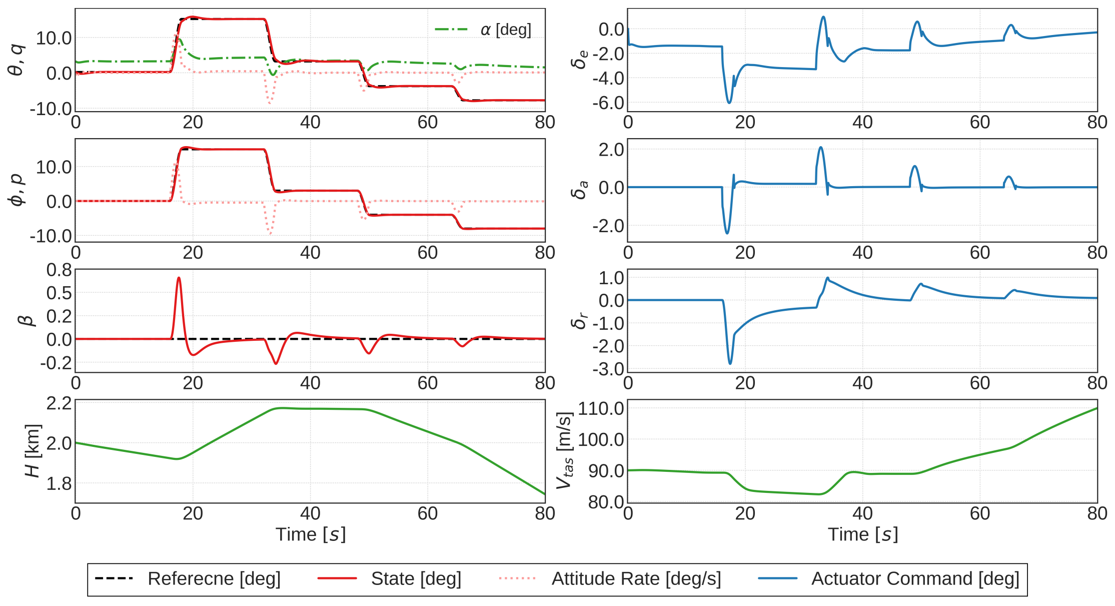
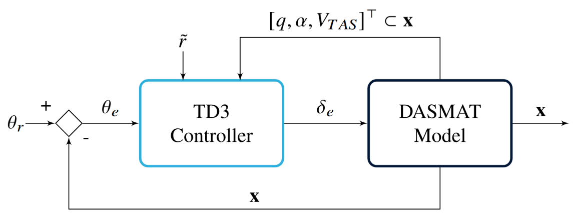
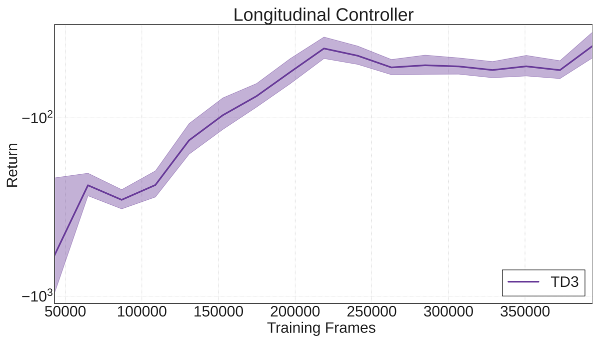


**Figure 9.2:** PID controllers for the pitch, roll and sideslip channels.



**(a)** Control diagram for pitch tracking.



**(b)** Learning curve.

This behaviour is expected as RLS, using a recursive linear model, is only accurate in the neighbourhood of the training samples. The RLS identification is thus verified only for short future predictions. Empirically, it has been estimated that prediction horizons shorter than $3\ s$ (coupled with frequent prediction times) are advisable.

## 9.5. Validation

After the testing is completed, validation would aim to prove that the offline-trained actors can provide attitude tracking on real-life aircraft systems.

First, the DASMAT framework used for simulation had already been validated by the work of [46]. Their experiments compared flight data obtained in the nominal pre-stall envelope with the output of the simulation model. The results showed a relative root mean squared error (RMSE) of $8.38\%$ and $12.65\%$ for longitudinal force and moment coefficients and $7.34\%$ and $8.58\%$ for the lateral ones. Thus, the modelled representation within the DASMAT framework was validated for flight conditions that belong to the pre-stall envelope. In the current thesis, the safety-informed genetic mutation specifically encourages the agent to not explore outside this envelope.

Second, the robustness and fault tolerance tests raise confidence in the controller's ability to deal with real-life scenarios. Despite this, training and testing only considered a limited number of faults and flight conditions. Moreover, the learning algorithms made the following simplifying assumptions: action selection and a partially observable MDP, no delays in actuator commands and sensor readings, ideal sensor fusion, aircraft remaining in nominal and clean configuration during training and the idealised atmospheric

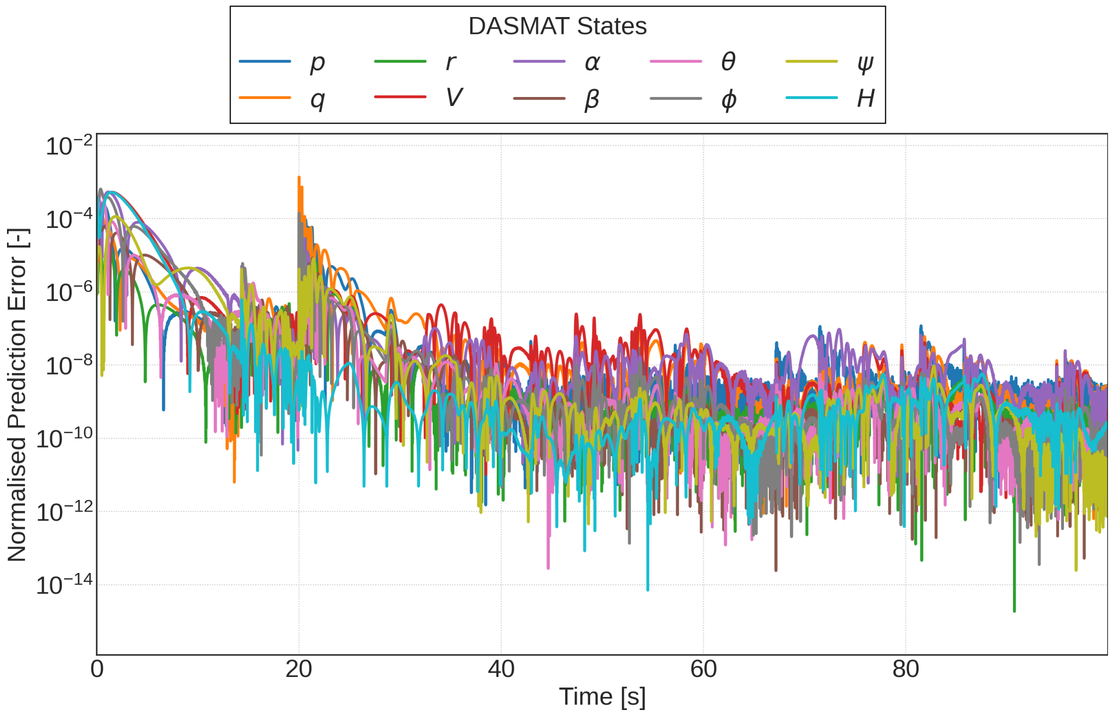


**Figure 9.4:** The normalised innovation of the RLS predictions. The model is updated continuously throughout the episode and the aircraft CG is suddenly shifted aft at $t = 20\ s$.

disturbance model.

Therefore, the results perform a necessary but unfortunately insufficient step towards validation. Whereas they cannot and should not replace flight testing, such campaigns are costly and time-consuming and, for fault-tolerance test cases, come with high safety risks. Ensuring a certain degree of control robustness is thus needed before the learned controllers can be transferred to the Flight Control Computer of real aircraft.

Validating the developed controller on a real-hardware task brings the extra challenges associated with the gap between simulation and reality. Bridging this gap would definitely be informative but such validation ultimately falls out of the scope of the present work and thus remains a highly recommended step for future research.

# Part IV

## Closure

# 10 Conclusion

## 10.1. Closing Remarks

The present work combined two bio-inspired frameworks, DRL and EA, culminating in SERL, a safety-informed ERL method that trains flight controllers on a non-linear model for a fixed-wing jet aircraft to provide optimal attitude tracking. It used a DDPG-based actor-critic structure with a GA loop. Extending the RL agent within PDERL, the TD3 algorithm improves sample efficiency and stability of the learning. The GA loop is a novelty-seeking crossover operator and safety-informed mutation that uses high-level domain knowledge about the flight envelope. With them, it obtains parameter-space exploration adding to the off-policy state space exploration of TD3.

The current work empirically proved that, by adjusting the population size and then selecting the adequate training time, SERL can achieve the desired balance between performance and control smoothness. Moreover, off-policy exploration using unregularised multi-layer neural networks was shown to aggravate the action noise phenomenon in simulation environments which only prioritise tracking performance.

Training a large controller population required proportionally more samples but achieved significantly higher fault tolerance than the TD3 agent in five out of six cases. It also remained robust to flight conditions and external disturbances in the presence of noise. This demonstrated the benefit of evolving a diverse population of controllers through distillation crossover and safety-informed mutations. Nevertheless, when system faults collapse the offline-obtained diversity, the TD3 remains marginally superior.

Directly balancing the performance-smoothness trade-off and benefiting from its proven robustness, the offline trained SERL agents can be more flexibly deployed on real-hardware applications. This represents a step towards making artificial intelligence a safety net for autonomous fault-tolerant controllers for safety-critical systems in general and aircraft in particular.

## 10.2. Research Questions

The current research thesis aimed to improve the robustness of fault-tolerant intelligent flight control by adapting a hybrid evolutionary reinforcement learning framework to provide reference-tracking control for a high-fidelity model of a small fixed-wing jet aircraft. Subsequently, it has answered a subset of the research questions proposed in the introductory chapter.

**Met-1**

What are the requirements for intelligent fault-tolerant flight control?

**Met-1.a** What are the most critical faults a modern flight control system shall adapt to?
**Met-1.b** What are the limitations of intelligent flight controllers in dealing with system faults?

Question **Met-1.a** was covered by Section 3.1.2 which compiles a list of plausible fault cases that should be considered while validating the fault-tolerant controller. Then, the answer to sub-question **Met-1.b** comes from the rest of Chapter 3.1 which offered an overview of intelligent control and its advantages and by Section 3.2.4 which delved deeper into the state-of-the-art class of deep reinforcement learning. Thus, it can be concluded that intelligent fault-tolerant controllers require a generalisable policy. In practice, this can only be achieved by finding a stable balance between safe exploration and exploitation of either domain-specific knowledge (e.g., modelled dynamics in online learning) or extensive series of previous experiences (e.g., the replay buffer or genetic memory in off-policy learning).

**Met-2**

What are the state-of-the-art offline hybrid methods combining deep reinforcement learning and evolutionary algorithms?

**Met-2.a** What are the strengths and limitations of deep reinforcement learning?
**Met-2.b** What are the strengths and limitations of evolutionary algorithms?
**Met-2.c** Which hybrid methods are proposed in the field's literature?
**Met-2.d** What benchmarks and corresponding metrics could be used to compare their control performance?

The second question within the Method Feasibility group was covered by Sections 3.2 to 3.4 from the Literature Study chapter. Namely, Chapter 3.2 focused on reinforcement learning, explaining its background, identifying the state-of-the-art and compiling its main advantages and disadvantages in the context of optimising flight control policies. Similarly, Chapter 3.3 treated evolutionary algorithms, introducing the concept of the quality-diversity search for repertoire learning. In contrast to reinforcement learning, evolutionary algorithms only update the policies based on completed episodes. Furthermore, they can be parallelised to reduced wall-clock training time and have a higher chance to explore outside of local optima.

Then, the two classes are combined in the hybrid framework of Evolutionary Reinforcement Learning. The historically-first ERL algorithm showed promising features but lacked stability during learning. This vulnerability prompted research work towards possible extensions. They were surveyed and reviewed to identify Proximal Distilled ERL as capable of stably evolving a population of control policies. The comparison was completed by using a worst-case performance metric that aims for high average rewards and low variance. Moreover, using explicit genetic operators and a DDPG actor-critic structure, the modular design of PDERL allows for straightforward updates and sensitivity studies.

**Met-3**

Which offline-trained hybrid framework is feasible to maximise the chosen metric for the control of a high-dimensional continuous system in the presence of faults?

Lastly, the feasibility of performing fault-tolerant control using the chosen hybrid framework was tested in Chapter 4. PDERL was trained offline on the LunarLander task from OpenAI gym, a toy problem domain in optimal continuous control. Compared to DDPG and CMA-ME, a state-of-the-art quality-diversity algorithm, PDERL showed better worst-case performance in both fault conditions considered: clipped actuator and unbiased noisy sensor observation. Moreover, the change in its behaviours was observed to be empirically correlated to the performance under the different faulty conditions. This verification completed the answer to question **Met-3** and thus finished the feasibility sub-set of the research questions.

Moving further from the preliminary analysis, the next research questions look at the implementation of the ERL framework as an intelligent fault-tolerant flight controller.

**Imp-1**

What is a suitable system/platform to train and test the hybrid flight controller?

**Imp-1.a** What simplified model and which initial conditions can be used to train the controller offline?
**Imp-1.b** What should be the architecture of the control loop, observation space and action space?
**Imp-1.c** Which fault conditions should be evaluated during testing?

The first question deals with the learning task and flight control architecture. Specifically, sub-question **Imp-1.a** was answered by employing DASMAT, a framework modelling non-linear 6-DOF fixed-wing aircraft dynamics framework as the simulation environment for learning and evaluation. The model was trimmed for steady straight symmetric flight at $H = 2000\ m$ and $V_{tas} = 90\ m/s$.

Then, the control task considered continuous attitude control following a random sequence of cosine-smoothed step references in pitch and roll while maintaining zero sideslip. The action space corresponded to the elevator, aileron and rudder deflection whereas the agent state was formed from the pitch, roll and yaw rate, angle of attack and tracking errors.

For evaluation, six fault cases were isolated from the previous work in the field based on their relevance and possibility to be modelled. They were: icing on wings, aft-shift in the centre of gravity, saturated actuators (aileron and elevator), partially broken tail resulting in reduced elevator effectiveness and, last but not least, a completely jammed rudder.

**Imp-2**

How can an offline learnt set of policies be efficiently and safely updated for online control of the selected platform in presence of faults?

**Imp-2.a** What are the most promising ways to use safety information to improve average controller performance with respect to the baseline hybrid framework?

**Imp-2.b** What is a sample-efficient method to select a well-behaving policy during online control of the faulty system?

The thesis investigated how the baseline hybrid agent can be improved and how it can be efficiently used for online control. The first challenge, marked by sub-question **Imp-2.a**, was tackled by adopting the TD3 structure and by designing the safety-informed mutation operator. The first improves sample efficiency by limiting catastrophic forgetting of the RL policy, a claim already verified within the field's literature. The mutation operator used domain knowledge to discourage parameter-space exploration in possibly unsafe directions. The safety-informed mutation offers an opportunity to incorporate expert-level knowledge concerning the controlled system via the user-defined cost function. Using even preliminary knowledge of the flight envelope decreases, over one generation, the number of unsafe transitions. Long-term learning beneficially impacts performance by limiting the chance of catastrophic forgetting. This increases the confidence that the safety-informed ERL can more easily be transferred to other flight control systems.

Then, Chapter 8 offers an answer to sub-question **Imp-2.b** by providing a proof-of-concept for an online adaption method which employs a sample-efficient Recursive Least Squares online model identification. Then, the tool uses the identified incremental model to predict the return of each available policy over a receiving time horizon. The best-performing individual by a margin switches with the behavioural policy in a hard (direct copy) or soft (Polyak average) manner. Testing this tool on an evaluation case with a sudden aft-shift of the aircraft's centre of gravity, the adaption via selection improved the tracking performance and the soft switching also increased the smoothness of the control actions.

**Imp-3**

How does the hybrid approach compare to an intelligent control framework based on DRL in terms of performance under faulty conditions?

SERL framework was compared to a state-of-the-art DRL agent trained under the same conditions (number of training frames, seed, hardware, and the number of hyperparameter tuning sweeps). TD3 and two configurations of SERL, namely SERL(10) and SERL(50), were tested on ten scenarios: the nominal conditions (i.e., same as during training), six fault cases, two different trim conditions and, last but not least, against an external disturbance in the presence of realistic sensor noise. For each case, the performance had been compared using the episode nMAE averaged over the controlled states: pitch, roll and sideslip.

The SERL(50) champion significantly outperformed the TD3-only actor in all but one case. As for most cases, the identified best-behaving actor was different from the nominal case, the success of SERL(50) can be attributed to the diversity obtained by evolving a genetic population of non-linear controllers. TD3 outperformed both SERL frameworks in the jammed rudder scenario when the blocked actuator caused the genetic diversity achieved in the actuating space to collapse.

Finally, by adjusting the population size and then selecting the adequate training time, SERL can balance performance and control smoothness. In contrast, TD3 showed action smoothness values lower by at least one order of magnitude than the SERL(50) champion. Thus, off-policy exploration using unregularised multi-layer neural networks was shown to aggravate the action noise phenomenon in simulation environments which only prioritise tracking performance.

## 10.3. General Contribution

With the increasing popularity of artificial intelligence, research in control systems has the challenge and opportunity to adapt and synchronise its state-of-the-art algorithms with novel bio-inspired intelligence methods.

Directly balancing the performance-smoothness trade-off and benefiting from its proven robustness, the offline trained SERL agents can be more flexibly deployed on real-hardware applications. In doing so, they perform a valuable step towards including artificial intelligence algorithms within autonomous fault-tolerant controllers, making them a robust safety net applicable to safety-critical systems in general and aircraft in particular.

# 11 Recommendations

Despite its achievements, the present work still faces limitations. This chapter provides a brief overview of the primary recommendations for the future continuation of this research project.

**Hyperparameter tuning**
Whereas this work considered limited hyperparameter tuning, this area should be explored in future iterations of the study. Specifically, more hyperparameter sweeps should be performed for a larger set of genetic population sizes.

**Improve sample-efficiency**
The hybrid methods suffer from poor sample efficiency. Ideally, such issues could be targeted by improving the sample efficiency of either or both learning frameworks. On one side, the work of [113] shows the opportunity to increase the training stability and efficiency of the RL agent by involving distributional critics. Potentially, this can benefit large population searches which hinder the RL convergences. On the other side, novel evolutionary strategy algorithms, such as Covariance Matrix Adaptation from [34], can replace the GA loop. For these extensions, the control policy smoothness and the possibility to add it to the fitness function shall be closely investigated.

**Improve online selection of actors**
The online adaption method can be improved by including means of uncertainty quantification over the prediction horizon. Also, the offline population can be tested against a set of safety constraints to identify and remove potentially unsafe policies. Lastly, scaling up the complexity of the identified model can in turn downgrade the sample efficiency. To prevent this from affecting the adaption, the agent can employ a probabilistic planner such as Monte Carlo Tree Search [33] for faster decision-making.

**6-DOF flight control**
This work only targeted the attitude control task and trained in one flight condition. Previous works have shown the opportunity to train 6-DOF controllers in a hierarchical manner. Achieving smooth control via direct actuator commands and benefiting from episodic evolutionary learning, the present work opens a possibility to circumvent the curse of dimensionality of 6-DOF control. Moreover, training such a controller on multiple points sampled within the flight envelope can increase its performance in the low dynamic pressure regime.

**Flight-test validation**
Proving robustness to changes in flight conditions, system and sensor dynamics and external disturbances, the work made valuable steps towards showing that SERL can provide attitude control on real systems. Despite this, validating it requires flight-testing campaigns. Generally, the problem of validating neural networks as flight control policies is cumbersome due to their black-box nature, which lacks explainability. Despite this, this necessary step would contribute towards their certification as flight controllers for future autonomous systems.

# References

[1] *Safety Report*. Tech. rep. International Civil Aviation Organization (ICAO), 2020. URL: `https://www.icao.int/safety/iStars/Pages/Accident-Statistics.aspx`.

[2] Peng Lu et al. "Aircraft fault-tolerant trajectory control using Incremental Nonlinear Dynamic Inversion". In: *Control Engineering Practice* 57 (2016), pp. 126–141.

[3] Christopher Edwards et al. "Fault tolerant flight control". In: *Lecture Notes in Control and Information Sciences* 399 (2010), pp. 1–560.

[4] Eugene Lavretsky et al. "Robust adaptive control". In: *Robust and adaptive control*. Springer, 2013, pp. 317–353.

[5] Richard S Sutton et al. *Reinforcement learning: An introduction*. MIT press, 2018.

[6] Bo Sun et al. "Incremental model-based global dual heuristic programming for flight control". In: *IFAC-PapersOnLine* 52.29 (2019), pp. 7–12.

[7] Y Zhou et al. "Incremental approximate dynamic programming for nonlinear flight control design". In: *Proceedings of the 3rd CEAS EuroGNC: Specialist Conference on Guidance, Navigation and Control, Toulouse, France, 13-15 April 2015*. 2015.

[8] S Sieberling et al. "Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction". In: *Journal of guidance, control, and dynamics* 33.6 (2010), pp. 1732–1742.

[9] Pieter Abbeel et al. "An Application of Reinforcement Learning to Aerobatic Helicopter Flight". In: *Advances in Neural Information Processing Systems*. Ed. by B. Scholkopf et al. Vol. 19. MIT Press, 2006. URL: `shorturl.at/clpY6`.

[10] Ian Goodfellow et al. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[11] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[12] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[13] Scott Fujimoto et al. "Addressing function approximation error in actor-critic methods". In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[14] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[15] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[16] Antonios Tsourdos et al. "Developing flight control policy using deep deterministic policy gradient". In: *2019 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*. IEEE. 2019, pp. 1–7.

[17] Minkyu Choi et al. "Soft Actor-Critic with Inhibitory Networks for Retraining UAV Controllers Faster". In: *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2022, pp. 1561–1570.

[18] Killian Dally et al. "Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control". In: *AIAA SCITECH 2022 Forum*. 2022, p. 2078.

[19] Eivind Bøhn et al. “Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization”. In: *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2019, pp. 523–533.

[20] Siddharth Mysore et al. “Regularizing action policies for smooth control with reinforcement learning”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 1810–1816.

[21] Antoine Cully et al. “Robots that can adapt like animals”. In: *Nature* 521.7553 (2015), pp. 503–507.

[22] Stephane Doncieux et al. “Evolutionary Robotics: What, Why, and Where to”. In: *Frontiers in Robotics and AI* 2 (2015). DOI: `10.3389/frobt.2015.00004`.

[23] Mehrdad Dianati et al. *An introduction to genetic algorithms and evolution strategies*. Tech. rep. University of Waterloo, Waterloo, Ontario, Canada, 2002.

[24] Evgenia Papavasileiou et al. “A Systematic Literature Review of the Successors of “NeuroEvolution of Augmenting Topologies””. In: *Evolutionary Computation* 29.1 (2021), pp. 1–73.

[25] Kenneth O Stanley et al. “Designing neural networks through neuroevolution”. In: *Nature Machine Intelligence* 1.1 (2019), pp. 24–35.

[26] Kenneth O Stanley et al. “Evolving neural networks through augmenting topologies”. In: *Evolutionary computation* 10.2 (2002), pp. 99–127.

[27] Kenneth O Stanley et al. “Efficient reinforcement learning through evolving neural network topologies”. In: *Proceedings of the 4th Annual Conference on genetic and evolutionary computation*. 2002, pp. 569–577.

[28] Joel Lehman et al. “Abandoning objectives: Evolution through the search for novelty alone”. In: *Evolutionary computation* 19.2 (2011), pp. 189–223.

[29] Justin K Pugh et al. “Quality diversity: A new frontier for evolutionary computation”. In: *Frontiers in Robotics and AI* 3 (2016), p. 40.

[30] Antoine Cully et al. “Quality and diversity optimization: A unifying modular framework”. In: *IEEE Transactions on Evolutionary Computation* 22.2 (2017), pp. 245–259.

[31] Bryan Lim et al. “Dynamics-aware quality-diversity for efficient learning of skill repertoires”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 5360–5366.

[32] Rituraj Kaushik et al. “Adaptive prior selection for repertoire-based online adaptation in robotics”. In: *Frontiers in Robotics and AI* (2020), p. 151.

[33] Konstantinos Chatzilygeroudis et al. “Reset-free trial-and-error learning for robot damage recovery”. In: *Robotics and Autonomous Systems* 100 (2018), pp. 236–250.

[34] Nikolaus Hansen. “The CMA evolution strategy: a comparing review”. In: *Towards a new evolutionary computation* (2006), pp. 75–102.

[35] Tim Salimans et al. “Evolution strategies as a scalable alternative to reinforcement learning”. In: *arXiv preprint arXiv:1703.03864* (2017).

[36] Maxime Allard et al. “Hierarchical Quality-Diversity for Online Damage Recovery”. In: *arXiv preprint arXiv:2204.05726* (2022).

[37] Olivier Sigaud. “Combining Evolution and Deep Reinforcement Learning for Policy Search: a Survey”. In: *arXiv preprint arXiv:2203.14009* (2022).

[38] J. H. Holland et al. “Cognitive systems based on adaptive algorithms”. In: *Pattern directed inference systems*. Ed. by D. A. Waterman et al. New York: Academic Press, 1978, pp. 313–329.

[39] Shimon Whiteson et al. “Sample-efficient evolutionary function approximation for reinforcement learning”. In: *AAAI*. 2006, pp. 518–523.

[40] Shimon Whiteson et al. “On-Line Evolutionary Computation for Reinforcement Learning in Stochastic Domains”. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Compu-*

*tation*. Seattle, Washington, USA: Association for Computing Machinery, 2006, pp. 1577–1584. DOI: 10.1145/1143997.1144252.

[41] Jörg KH Franke et al. “Sample-efficient automated deep reinforcement learning”. In: *arXiv preprint arXiv:2009.01555* (2020).

[42] Shauharda Khadka et al. “Evolution-guided policy gradient in reinforcement learning”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[43] Cristian Bodnar et al. “Proximal distilled evolutionary reinforcement learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 3283–3290.

[44] Shauharda Khadka et al. “Collaborative evolutionary reinforcement learning”. In: *International conference on machine learning*. PMLR. 2019, pp. 3341–3350.

[45] Van Der Linden. “DASMAT-Delft University aircraft simulation model and analysis tool: A Matlab/Simulink environment for flight dynamics and control analysis”. In: *Series 03: Control and Simulation 03* (1998). URL: http://resolver.tudelft.nl/uuid:9bfb1f68-2f7c-4f32-91b4-79297d295f84.

[46] MA Van den Hoek et al. “Identification of a Cessna Citation II model based on flight test data”. In: *Advances in Aerospace Guidance, Navigation and Control*. Springer, 2018, pp. 259–277. URL: http://resolver.tudelft.nl/uuid:c0a57513-38b7-4d3a-898c-fa57c3e7ac2e.

[47] Peggy Williams-Hayes. “Flight test implementation of a second generation intelligent flight control system”. In: *Infotech@ Aerospace*. 2005, p. 6995.

[48] Huaguang Zhang et al. “Adaptive Fuzzy Fault-Tolerant Tracking Control for Partially Unknown Systems With Actuator Faults via Integral Reinforcement Learning Method”. In: *IEEE Transactions on Fuzzy Systems* 27.10 (2019), pp. 1986–1998. DOI: 10.1109/TFUZZ.2019.2893211.

[49] Frank Lynch et al. “Effects of ice accretions on aircraft aerodynamics”. In: *Progress in Aerospace Sciences - PROG AEROSP SCI* 37 (Nov. 2001), pp. 669–767. DOI: 10.1016/S0376-0421(01)00018-5.

[50] Peng Lu. “Fault diagnosis and fault-tolerant control for aircraft subjected to sensor and actuator faults”. PhD thesis. Technische Universiteit Delft, 2016.

[51] Earl H Mckinney Jr et al. “How swift starting action teams get off the ground: What United flight 232 and airline flight crews can tell us about team communication”. In: *Management Communication Quarterly* 19.2 (2005), pp. 198–237.

[52] Dimitri P Bertsekas et al. *Neuro-dynamic programming*. Athena Scientific, 1996.

[53] Dimitri Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[54] Richard Bellman et al. *Dynamic programming and modern control theory*. Vol. 81. Elsevier Science, 1965.

[55] Karl Johan Åström. “Optimal control of Markov processes with incomplete state information”. In: *Journal of mathematical analysis and applications* 10.1 (1965), pp. 174–205.

[56] Alois Pourchot et al. “CEM-RL: Combining evolutionary and gradient-based methods for policy search”. In: *arXiv preprint arXiv:1810.01222* (2018).

[57] Lucian Busoniu et al. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2017.

[58] Danil V Prokhorov et al. “Adaptive critic designs”. In: *IEEE transactions on Neural Networks* 8.5 (1997), pp. 997–1007.

[59] Carl Andersson. “Deep learning applied to system identification: A probabilistic approach”. PhD thesis. Uppsala University, 2019.

[60] Djork-Arné Clevert et al. “Fast and accurate deep network learning by exponential linear units (elus)”. In: *arXiv preprint arXiv:1511.07289* (2015).

[61] Hong Qian et al. “Derivative-free reinforcement learning: a review”. In: *Frontiers of Computer Science* 15.6 (2021), pp. 1–19.

[62] Xavier Glorot et al. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[63] Diederik P Kingma et al. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[64] Felipe Petroski Such et al. “Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning”. In: *arXiv preprint arXiv:1712.06567* (2017).

[65] Jiuxiang Gu et al. “Recent advances in convolutional neural networks”. In: *Pattern Recognition* 77 (2018), pp. 354–377.

[66] Larry Medsker et al. *Recurrent neural networks: design and applications*. CRC press, 1999.

[67] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[68] John D Owens et al. “GPU computing”. In: *Proceedings of the IEEE* 96.5 (2008), pp. 879–899.

[69] Reza Refaei Afshar et al. “Automated Reinforcement Learning: An Overview”. In: *arXiv preprint arXiv:2201.05000* (2022).

[70] Nicolai A Lynnerup et al. “A survey on reproducibility by evaluating deep reinforcement learning algorithms on real-world robots”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 466–489.

[71] *OpenAI Spinning Up*. `https://spinningup.openai.com/en/latest/index.html`. Accessed: 2022/05/04.

[72] Marcin Andrychowicz et al. “Hindsight experience replay”. In: *Advances in neural information processing systems* 30 (2017).

[73] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3 (1992), pp. 229–256.

[74] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[75] Boris T Polyak. “Some methods of speeding up the convergence of iteration methods”. In: *Ussr computational mathematics and mathematical physics* 4.5 (1964), pp. 1–17.

[76] Sebastian Thrun et al. “Issues in using function approximation for reinforcement learning”. In: *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*. Vol. 6. 1993.

[77] Tommaso Mannucci et al. “Hierarchically Structured Controllers for Safe UAV Reinforcement Learning Applications”. In: Jan. 2017. DOI: `10.2514/6.2017-0791`.

[78] Tijmen Pollack et al. “Safe Curriculum Learning for Optimal Flight Control of Unmanned Aerial Vehicles with Uncertain System Dynamics”. In: *AIAA Scitech 2020 Forum*. 2020, p. 2100.

[79] John R Koza et al. “Genetic programming”. In: *Search methodologies*. Springer, 2005, pp. 127–164.

[80] Edgar Galván et al. “Neuroevolution in deep neural networks: Current trends and future challenges”. In: *IEEE Transactions on Artificial Intelligence* 2.6 (2021), pp. 476–493. DOI: `10.1109/TAI.2021.3067574`.

[81] Marco A Wiering et al. “Reinforcement learning: State-of-the-art”. In: *Adaptation, learning, and optimization* 12.3 (2012), p. 729.

[82] Byoung-Tak Zhang et al. “Evolving optimal neural networks using genetic algorithms with Occam’s razor”. In: *Complex systems* 7.3 (1993), pp. 199–220.

[83] Ajay Shrestha et al. “Improving genetic algorithm with fine-tuned crossover and scaled architecture”. In: *Journal of Mathematics* 2016 (2016).

[84] Ilya Loshchilov et al. “CMA-ES for hyperparameter optimization of deep neural networks”. In: *arXiv preprint arXiv:1604.07269* (2016).

[85] Manolis Papadrakakis et al. “Structural optimization using evolution strategies and neural networks”. In: *Computer methods in applied mechanics and engineering* 156.1-4 (1998), pp. 309–333.

[86] Christian Igel et al. “Covariance matrix adaptation for multi-objective optimization”. In: *Evolutionary computation* 15.1 (2007), pp. 1–28.

[87] Matthew C Fontaine et al. “Covariance matrix adaptation for the rapid illumination of behavior space”. In: *Proceedings of the 2020 genetic and evolutionary computation conference*. 2020, pp. 94–102.

[88] Jean-Baptiste Mouret et al. “Illuminating search spaces by mapping elites”. In: *arXiv preprint arXiv:1504.04909* (2015).

[89] Konstantinos Chatzilygeroudis et al. “Quality-Diversity Optimization: a novel branch of stochastic optimization”. In: *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*. Springer, 2021, pp. 109–135.

[90] Andrés Arias et al. “An IEEE Xplore database literature review regarding the interaction between electric vehicles and power grids”. In: *2015 IEEE PES Innovative Smart Grid Technologies Latin America (ISGT LATAM)* (2015), pp. 673–678.

[91] Antoine Cully. “Multi-emitter MAP-elites: improving quality, diversity and data efficiency with heterogeneous sets of emitters”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021, pp. 84–92.

[92] Bryan Lim et al. “Accelerated Quality-Diversity for Robotics through Massive Parallelism”. In: *arXiv preprint arXiv:2202.01258* (2022).

[93] Leon Keller et al. “Model-based quality-diversity search for efficient robot learning”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 9675–9680.

[94] Bryon Tjanaka et al. “Approximating Gradients for Differentiable Quality Diversity in Reinforcement Learning”. In: *arXiv preprint arXiv:2202.03666* (2022).

[95] Geoffrey Cideron et al. “Qd-rl: Efficient mixing of quality and diversity in reinforcement learning”. In: *arXiv preprint arXiv:2006.08505* (2020).

[96] Norman P Jouppi et al. “In-datacenter performance analysis of a tensor processing unit”. In: *Proceedings of the 44th annual international symposium on computer architecture*. 2017, pp. 1–12.

[97] Jose Antonio Martin H et al. “Learning autonomous helicopter flight with evolutionary reinforcement learning”. In: *International Conference on Computer Aided Systems Theory*. Springer. 2009, pp. 75–82.

[98] Simyung Chang et al. “Genetic-gated networks for deep reinforcement learning”. In: *Advances in neural information processing systems* 31 (2018).

[99] Emanuel Todorov et al. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, pp. 5026–5033.

[100] Enrico Marchesini et al. “Exploring safer behaviors for deep reinforcement learning”. In: *AAAI Conference on Artificial Intelligence*. 2021.

[101] Federico Espositi et al. “Gradient Bias to Solve the Generalization Limit of Genetic Algorithms Through Hybridization with Reinforcement Learning”. In: *International Conference on Machine Learning, Optimization, and Data Science*. Springer. 2020, pp. 273–284.

[102] Karush Suri et al. “Maximum mutation reinforcement learning for scalable control”. In: *arXiv preprint arXiv:2007.13690* (2020).

[103] Yan Ma et al. “Evolutionary Action Selection for Gradient-based Policy Learning”. In: *arXiv preprint arXiv:2201.04286* (2022).

[104] Yuxing Wang et al. “A Surrogate-Assisted Controller for Expensive Evolutionary Reinforcement Learning”. In: *arXiv preprint arXiv:2201.00129* (2022).

[105] Andrei A Rusu et al. “Policy distillation”. In: *arXiv preprint arXiv:1511.06295* (2015).

[106] Joel Lehman et al. “Safe mutations for deep and recurrent neural networks through output gradients”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2018, pp. 117–124.

[107] Javier Garcıa et al. “A comprehensive survey on safe reinforcement learning”. In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480.

[108] Casper Teirlinck. *Reinforcement Learning for Flight Control: Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance*. Tech. rep. Delft University of Technology, 2022. URL: `http://resolver.tudelft.nl/uuid:dae2fdae-50a5-4941-a49f-41c25bea8a85`.

[109] David J Moorhouse et al. *Background information and user guide for MIL-F-8785C, military specification-flying qualities of piloted airplanes*. Tech. rep. Air Force Wright Aeronautical Labs Wright-Patterson AFB OH, 1982.

[110] Fabian Grondman et al. “Design and flight testing of incremental nonlinear dynamic inversion-based control laws for a passenger aircraft”. In: *2018 AIAA Guidance, Navigation, and Control Conference*. 2018, p. 0385.

[111] Ye Zhou et al. “Nonlinear adaptive flight control using incremental approximate dynamic programming and output feedback”. In: *Journal of Guidance, Control, and Dynamics* 40.2 (2016), pp. 493–496.

[112] Guillaume Chaslot et al. “Monte-carlo tree search: A new framework for game ai”. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 4. 1. 2008, pp. 216–217.

[113] Peter Seres et al. *Distributional Reinforcement Learning for Flight Control: A risk-sensitive approach to aircraft attitude control using Distributional RL*. Tech. rep. Delft University of Technology, 2022. URL: `http://resolver.tudelft.nl/uuid:6cd3efd1-b755-4b04-8b9b-93f9dabb6108`.