Exploration-Driven Reinforcement Learning for Avionic System Fault Detection (Experience Paper)

Paul-Antoine Le Tolguenec, Emmanuel Rachelson, Yann Besse, Florent Teichteil-Koenigsbuch, Nicolas Schneider, Hélène Waeselynck, Dennis Wilson

Exploration-Driven Reinforcement Learning for Avionic System Fault Detection (Experience Paper)

Paul-Antoine Le Tolguenec
ISAE SUPAERO
France
paul-antoine.le-tolguenec@isae-supaero.fr

Emmanuel Rachelson
ISAE SUPAERO
France
Emmanuel.RACHELSON@isae-supaero.fr

Yann Besse
Airbus
France
yann.besse@airbus.com

Florent Teichteil-Koenigsbuch
Airbus
France
florent.teichteil-koenigsbuch@airbus.com

Nicolas Schneider
Airbus
France
nicolas.schneider@airbus.com

Hélène Waeselynck
LAAS-CNRS
France
helene.waeselynck@laas.fr

Dennis Wilson
ISAE SUPAERO
France
Dennis.WILSON@isae-supaero.fr

Abstract

Critical software systems require stringent testing to identify possible failure cases, which can be difficult to find using manual testing. In this study, we report our industrial experience in testing a realistic R&D flight control system using a heuristic based testing method. Our approach utilizes evolutionary strategies augmented with intrinsic motivation to yield a diverse range of test cases, each revealing different potential failure scenarios within the system. This diversity allows for a more comprehensive identification and understanding of the system's vulnerabilities. We analyze the test cases found by evolution to identify the system's weaknesses. The results of our study show that our approach can be used to improve the reliability and robustness of avionics systems by providing high-quality test cases in an efficient and cost-effective manner.

CCS Concepts

• **Computer systems organization** → **Reliability**; • **Computing methodologies** → **Reinforcement learning**; • **Applied computing** → **Avionics**; • **Software and its engineering** → **Software testing and debugging**.

Keywords

Reinforcement learning, intrinsic motivation, genetic algorithms, evolutionary strategies, diversity, software reliability, physical system, critical software system, automated testing

ISSTA '24, September 16–20, 2024, Vienna, Austria

ACM ISBN 979-8-4007-0612-7/24/09
https://doi.org/10.1145/3650212.3680331

ACM Reference Format:
Paul-Antoine Le Tolguenec, Emmanuel Rachelson, Yann Besse, Florent Teichteil-Koenigsbuch, Nicolas Schneider, Hélène Waeselynck, and Dennis Wilson. 2024. Exploration-Driven Reinforcement Learning for Avionic System Fault Detection (Experience Paper). In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3650212.3680331

1 Introduction

High-assurance systems must be designed to tolerate faults and prevent the occurrence of critical failures. The architectural solutions are based on redundancy and on the introduction of detection and protection mechanisms. While such solutions are necessary to meet dependability requirements, they add complexity to the system, which introduces new risks. There may be cases where dependability mechanisms cause system failures that would not exist without them. The problem may be illustrated by a first example from the railway domain. In Essame et al. [19], the authors addressed a scenario involving redundant units to control a section of railway track, in a primary/secondary configuration. In the studied scenario, the primary unit fails and control is passed to the secondary one. However, due to an inconsistency between the two units (the primary has detected a new train that the other has not yet seen), the handover operation ends with an unregistered train, making the system unable to ensure the no-collision property. Interestingly, in this architecture, each unit is fail-safe: no collision could occur if the unit was alone. It is the improper management of redundancy that creates the safety issue. A second example of failure comes from false positives of detection mechanisms. For instance, in the automotive domain, self-driving vehicles are reported to suffer from spurious emergency braking, a phenomenon known as "phantom braking". It has been a persistent issue for Tesla vehicles and is currently under investigation by the U.S. safety regulators [1].

In this paper, we report on the search for false positives in an avionic system, yielding the unexpected disengagement of automated control functions. This case study uses the command (COM) and monitoring (MON) COM/MON architecture typical of Airbus systems [59], which uses command and monitoring units to independently compute orders to send to aircraft actuators, allowing for comparison between the two units. When a disagreement is detected, the units are reported as faulty. The units may have a different view of the physical environment (like in the previous example in the railway domain), due to asynchrony. As such, a small discrepancy in the computed values is accepted. However, there is a risk for corner cases, where the units are properly functioning and yet the discrepancy exceeds the detection threshold. Finding such corner cases is considered as very difficult by test engineers. Formal methods may not help, because the problem cumulates many aspects that would make formal analysis challenging. From a study on model checking at Airbus, these aspects are the following [9]: (i) the multiprocessor architecture, which requires accounting for communication delays and clocks asynchrony, (ii) the multiple computation periods inside a unit, different pieces of code being cyclically executed at different periodicity, (iii) the numerical floating point calculation. Hence, testing methods are more realistic candidates. Ideally, the test method should aid the engineer in their ability to: identify potential corner cases if they exist, categorize them if multiple exist, and even characterize other dangerous cases where the system is close to fail but does not.

Our study investigates the potential effectiveness of new Reinforcement Learning (RL) algorithms to meet these needs. More specifically, our study focuses on the use of the recently published Curiosity-ES algorithm [31], which is a policy search algorithm designed to tackle Reinforcement Learning problems where exploration is crucial. This algorithm enhances the agent's ability to explore its environment by incorporating a curiosity-driven component, encouraging the agent to seek out novel states. The contributions of this paper can be summarized as follows:

- A study on identifying failure cases in a realistic COM/MON industrial system, developed by an R&D department.
- The formalization of searching for these failure cases as a sequential decision-making problem.
- The comparison between different baselines which highlight the benefits of using highly exploratory approaches such as Curiosity-ES.
- A detailed discussion of the results obtained and an identification of the key features leading to disagreement.
- The application of a statistical post-processing technique to categorize generated failure cases into clusters, enhancing a finer analysis of their diversity.

Our implementations are available in the replication package.

2 Related Work

In this part, we explore the existing literature on automated software testing in different critical systems. More specifically, We mostly focus on the methodologies used to test reactive systems that continuously interact with their environment. These systems are typically tested in X-in-the-loop configurations, where X can represent software, hardware, or model elements. In the pursuit to uncover corner cases within these systems, there has been a significant reliance on heuristic methods, which employ experience-based techniques for problem-solving and decision-making. These methods are particularly valued for their ability to navigate the complex, often unpredictable nature of reactive systems, guiding the testing process toward scenarios that might reveal hidden vulnerabilities. In this section, we will focus on two major categories of these methods: "Sample-based Testing" approaches and "Reinforcement Learning" techniques, beginning with the former.

Search-based testing approaches often treat the test sequence as a whole, meaning that each test scenario is conditioned by a set of parameters. It is these parameters that are optimized to approach a failure case and expose a flaw if one exists. These methods require to define a 'fitness' score for each test case and this fitness is maximized by employing an evolutionary algorithm to sample the parameters adaptively. This fitness can often be understood as the opposite of a distance between the executed scenario and a potential failure case. Choosing the right fitness function is critical for testing success, but it can be challenging and small changes on this function can greatly impact results. McMinn [42] provides an overview of various search-based testing approaches that have been successfully used in various areas such as temporal testing [48, 61–63], functional testing [11, 62] and structural testing [3, 58, 60], including concrete examples of fitness functions employed. Recently, the field of cyber-physical systems has delved deeply into these methods, especially for evaluating the robustness of autonomous vehicles. [4, 26, 37, 54, 55]. Recent advancements include using adaptive importance sampling and Bayesian optimization in autonomous vehicle software to identify and analyze potential critical system failures [2, 46], offering deeper insights into system behaviors and vulnerabilities. Moreover, when searching for optimal parameters, these heuristic search-based approaches are known for their need to maintain a balance between exploration and exploitation. The evolutionary community has thoroughly investigated the question of massive exploration and the balance with exploitation through concepts like Novelty Search[33–35] and Quality Diversity [16, 17]. This idea has been applied in search-based testing, and numerous studies have supported its relevance [10, 22, 41]. These approaches hold even greater value as it has been tested and proven in real-world industrial contexts [21, 40]. The limiting aspect of search-based testing approaches is that by treating the testing sequence as a whole they are not designed to react sensitively to every system behavior, but rather to effectively explore a predefined parameter space. An alternative to these approaches is to frame the testing problem as a sequential decision-making one. This formulation then allows for the application of Reinforcement Learning algorithms (RL). Reinforcement Learning [49, 57](RL) is a subset of machine learning wherein an agent optimizes its policy/strategy through interactions with an environment, aimed at achieving specific objectives. The learning mechanism is driven by feedback, manifesting as rewards or penalties, contingent on the actions taken, rather than direct instruction. This iterative process allows the agent to formulate a policy that incrementally maximizes the total expected reward over time. The concept of using RL to identify flaws in systems was recently formalized as "Adaptive Stress Testing" by [27, 28, 32]. The approach views the link between the system under test and the physical environment simulated as the environment

itself, with the agent's actions introducing disturbances in that loop. This formulation enables the use of the variety of RL algorithms; for example, in a discrete state space environment, well-understood methods like Q-Learning can be particularly effective as they provide theoretical guarantees that the strategy found is the best for a given criteria. These methods have been widely used to assess the robustness of various cyber-physical systems such as the transmission system, power system with automatic voltage controls, or even to asses a system availability against DoS attacks (Distributed Denial of Service) [13, 18, 30, 36, 39, 64, 65]. In complex systems where simple state abstraction isn't viable, Deep Reinforcement Learning uses neural networks as function approximators to determine the optimal policy [44]. The study by Liu et al. [38] provides a comprehensive look at how these methods are applied in Cyber-Physical Systems (CPS), particularly in the transportation and industrial fields. Recently, [43] further explored DRL for uncovering software defects in various CPS . [20] developed a specific experience replay method for efficiently training policies targeting faults in autonomous driving systems. Similar to evolutionary approaches, numerous methods stemming from deep reinforcement learning have been developed to extensively explore the state space of an environment. These methods are often based on an exploration bonus added to the reward [6, 12, 47]. This type of exploration technique has proven to be effective in recent testing applications[25, 66]. Finally, recently numerous studies have highlighted the benefits of hybridizing evolutionary strategies and Reinforcement Learning algorithms. Practically speaking, evolutionary approaches [50], whose evaluation is easily parallelized, can be used to optimize the parameters of neural networks themselves [14, 15, 52]. The advantage of using evolutionary approaches for parameter optimization, rather than action sequences as in Search-Based Testing (SBT), lies in the inductive bias capabilities of neural networks. Our study employs the recently published Curiosity-ES algorithm [31] that falls within this category of methods. This algorithm, which combines evolutionary strategies with a curiosity-driven exploration bonus, has demonstrated remarkable exploration capabilities across various benchmarks. We demonstrate the relevance of this method for searching false positive in a COM/MON system, which will be explained in detail in the following section.

3 Avionic System Testing: A Reinforcement Learning Perspective

3.1 Testing Architecture Overview

Our testing approach aims to uncover false positives in a simulated COM/MON system. The testing architecture is depicted in Figure 1. The simulator we used includes aircraft dynamics modeling, for which JSBSIM is employed, along with a model of the aircraft system itself. These two components interact in a closed-loop system. The simulator operates synchronously and deterministically. Temporal asynchrony between the two units is simulated by adjusting each unit's internal clock. Following the framing of Adaptive Stress Testing [32], we study the inclusion of an agent which interacts with this closed-loop system. At each timestep, this agent collects comprehensive state information of the system and physical state data

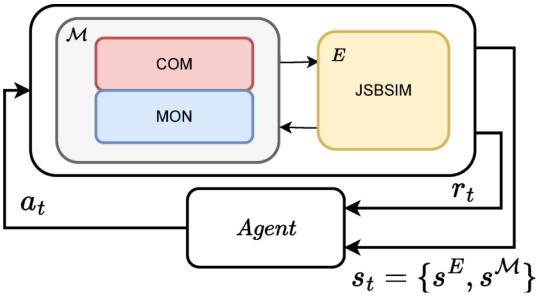

Figure 1: Test architecture overview

of the aircraft (JSBSIM state). Using this information, the agent can influence the closed-loop by altering the aircraft's control surface positions and the asynchrony between the two units by adjusting their clocks. The agent acts as an oracle, serving both as the pilot and controlling the system's asynchrony rate.

3.2 COM/MON Modeling

The COM/MON is an architecture which enables a constant monitoring of the orders sent by the EFCS (Electrical Flight Control System) to the aircraft surfaces controller. The COM computes a command to send to the controller based on data from the multiple sensors. The MON recomputes the order, based on its own inputs, but potentially sampled at a different time. The orders from the two units should be consistent, but if the COM/MON order difference is maintained above a certain threshold for enough time, the EFCS triggers a fault.

The COM and MON units have almost the same software and hardware architecture, although they may differ on some very specific points. The two units are almost fully independent and each uses an independent clock. This independence guarantees efficient monitoring, but it can also induce random (bounded) asynchronicity between the two units. This temporal shifting combined with certain events can produce an unwanted COM/MON disagreement leading to an EFCS fault. Identifying the spurious events which could cause COM/MON disagreement during the development of the unit logic is a challenging task due to the many confounding factors: the inputs from the different sensors, the command sent by the pilot, and the synchronization of the two units. In this article, we study the use of reinforcement learning to find such faults by reward disagreement between the COM and MON units.

We model the COM/MON architecture contained in the flight controller using a realistic version of a common aircraft system design. An overview of the architecture is presented in Figure 2. Specifically, we aim to test the part of the software which acts on the longitudinal evolution of the plane during the landing phase, hence the elevator position δ_q.

We refer to the physical environment as E and the system interacting with it as $\mathcal{M}$. $\mathcal{M}$ can be seen as a function which takes as input a pitch rate order $u_{\dot{\theta}}$, the actual pitch rate $\dot{\theta}$, an internal memory state, and a boolean b indicating impact, which is based on the compression of the front gear. The function $\mathcal{M}$ then outputs a position command on the elevator surface δ_q. The logic used

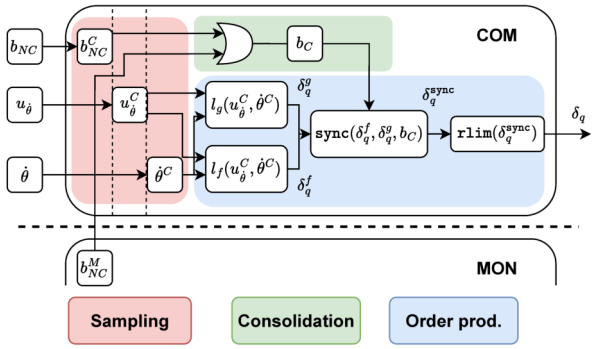

Figure 2: The COM architecture, which calculates control orders for the elevator surface δ_q. The MON architecture is equivalent and provides the binary variable b_{NC}^M to the COM.

(Flight/Ground) to compute the order of surface positions is conditioned by the impact boolean b_C, which indicates whether the plane has touched the ground or not. To ensure both units adhere to the same rule, a mechanism is in place to resynchronize them. The COM and MON units of $\mathcal{M}$ use different clock times t^C and t^M, for COM and MON respectively. The order calculated by the COM δ_q^C is the one used by the system $\mathcal{M}$. The MON recalculates the order and the disagreement between the COM and the MON is calculated as $\Delta_{\delta_q}^{C/M} = |\delta_q^C - \delta_q^M|$. The internal calculations of the COM/MON units are executed sequentially with distince frequencies per calculation. These calculations can be split into three groups: sampling, consolidation, and order production.

Sampling In each unit, the variables $u_{\dot{\theta}}$, $\dot{\theta}$, and b are sampled from E with a specific t_l and stored in memory. For example, every 10ms the COM samples $u_{\dot{\theta}}(t)$ and the MON samples $u_{\dot{\theta}}(t + \Delta_t^{C/M})$, where $\Delta_t^{C/M}$ is the phase shift between the two units. For a given unit n (C or M), we refer to the stored variable as $u_{\dot{\theta}}^n$, $\dot{\theta}^n$ and b_{NC}^n. NC stands for non-consolidated b, as explained just below.

Consolidation Although COM and MON are independent, the critical variable b is consolidated between the two units: $b_C = b_{NC}^n \,||\, b_{NC}^{op}$ where op indicates the opposite unit. Through this consolidation, both units are made aware of an impact, even if one unit has not yet been informed of the impact in the b_{NC}^n variable due to phase shift. We model the time delay of the communication between the units as $t_d \in [2.8, 3.2]$ms.

Order production Two orders, δ_q^f and δ_q^g, are generated using the flight and ground control laws, $l_f(u_{\dot{\theta}}, \dot{\theta})$ and $l_g(u_{\dot{\theta}}, \dot{\theta})$, respectively. These control laws are Proportional Integral Derivative (PID) controllers with different gains. The various gains are summarized here:

	Flight	Ground
K_p	5	5
K_i	10	0
K_d	0.01	0.01

It is important to note that these gains were specifically designed to reach an optimal response on the pitch rate order. The orders are then consolidated by a synchronization unit which is conditioned by b_C: $\text{sync}(\delta_q^f, \delta_q^g, b_C) = \delta_q^{\text{sync}}$. This consolidation unit is intended to produce a temporally linear transition between the two orders. Finally, the consolidated order is passed through a rate limiter, $\texttt{rlim}(\delta_q^{\text{sync}}) = \delta_q$, to ensure that it is 1-Lipschitz continuous with respect to time.

Each operation is executed at a frequency of time t_f, which is different per operation. For $u_{\dot{\theta}}$ and $\dot{\theta}$, t_f = 10ms. The order δ_q produced by $\mathcal{M}$ and given to E is refreshed every 40ms and the impact boolean b_{NC} is sampled with t_f = 120ms.

In this work, the environment E is an instance of JSBSim [8], which simulates a realistic flight pattern of the aircraft and the environment. It should be noted that in this context, $u_{\dot{\theta}}$ and b_{NC} are derived directly from the simulation without any noise or malfunction. Therefore, if a disagreement event arises, it is not due to a malfunction in an upstream computer. The dynamics used are those of a commercial aircraft. The flight scenario is shown in Figure 3 and represents a standard landing pattern with a slight bounce, depending on the pilot actions.

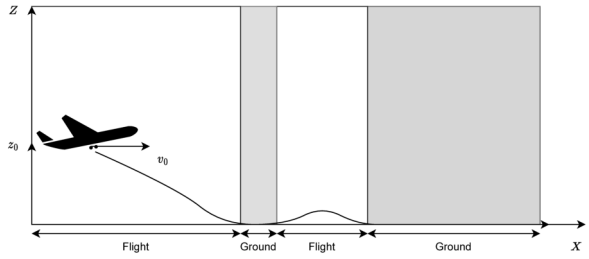

Figure 3: The flight scenario studied in this work. The aircraft is initialized in a landing pattern where bouncing is possible based on the given controls. The different activation phases of the flight control law l_f and ground control l_g are shown in background color.

3.3 Reinforcement Learning for Testing COM/MON Disagreement

Despite its simplicity, the system described can take on many states and physical conditions, making flaw detection a non-trivial task. We formulate the problem of finding COM/MON disagreement as a sequential decision-making problem, as in [27]. We configure the state space and transition model to create a fully observable Markov Decision Process [7]. Formally, we consider the process $\mathcal{E} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the transition function from each state to the next one depending on the action, and $\mathcal{R}$ is the reward function. We detail each component below.

For each time step t, the state $s_t \in \mathcal{S}$ contains both the state of the simulated airplane environment E and the state of the COM/MON units $\mathcal{M}$. Formally, we note that a state $s_t = \{s_t^E, s_t^{\mathcal{M}}\} \in S^E \times S^{\mathcal{M}}$, with S^E and $S^{\mathcal{M}}$ being respectively the set of states of the environment E and the system $\mathcal{M}$.

The state of the COM/MON system $s^{\mathcal{M}}$ contains all of the sampled variables, which are stored in the memory of the two units. In addition, it contains the state variables of the flight and ground

control laws l_f and l_g, the consolidator synchronization function sync, and the rate limiter function rlim. Finally, it also contains the current COM/MON desynchronization $\Delta_t^{C/M}$. The state of the environment s^E contains the variables which define the physical state of the aircraft, such as $\theta, \dot{\theta}, h, u_{\dot{\theta}}$. We note that the variables $\dot{\theta}$ and $u_{\dot{\theta}}$ are sampled by the COM and MON units; these variables are therefore present in both $s^{\mathcal{M}}$ and s^E. However, the sampled values in $s^{\mathcal{M}}$ are the most recently sampled values from the COM and MON, whereas the values in s^E are the current values of the simulated airplane. The total state space is of dimension 45: $s \in \mathbb{R}^{45}$. A reinforcement learning agent π is trained to find the optimal action $a_t = (\partial u_{\dot{\theta}(t)}, \partial \Delta_t^{C/M}) \in \mathcal{A}$, where $\partial u_{\dot{\theta}(t)}$ is a change on the pitch rate order and $\partial \Delta_t^{C/M}$ is a change of the desynchronization of the two units. To ensure that the agent only explores possible trajectories, we add physical constraints to the action space $\mathcal{A}$: $|\frac{du_{\dot{\theta}}}{dt}| < 100°/\text{s}$, $u_{\dot{\theta}} \in [-45°, 45°]$, $|\frac{d\Delta_t^{C/M}}{dt}| < 5\text{ms/s}$ and $\texttt{max}(|\Delta_t^{C/M}|) < 40$ ms. As such, there are six possible actions, three for $u_{\dot{\theta}}$ and three for $\Delta_t^{C/M}$:

$\partial u_{\dot{\theta}}$	$\partial \Delta_t^{C/M}$
+0.1°	$+5\mu s$
-0.1°	$-5\mu s$
0°	$0\mu s$

Every 1ms, an action for $u_{\dot{\theta}}$ and an action for $\Delta_t^{C/M}$ are taken using $\texttt{argmax}(a_t)$ over the possible actions for each type.

Our goal is to find feasible yet unlikely sequences of actions, or trajectories, which lead to the maximum disagreement between the COM and MON units. As such, the agent is rewarded based on the current disagreement between the two units $\mathcal{R}(s_t, a_t) = \Delta_{\delta_q}^{C/M}(t)$.

Once the agent has taken an action, the transition model $\mathcal{P}$ determines the next state. Specifically, we use the JBSim simulator of the environment and our simulation of the COM/MON model to calculate s_t. We configure the flight scenario shown in Figure 3 to follow a standard landing pattern. We use a PID to find the optimal command: $u_{\dot{\theta}}^{\text{opt}}(t)_{t \in \{0,\dots,H\}}$ H being the horizon of the evaluation (in our experiments, $H = 15$s.). The actual command used in the environment is therefore : $u_{\dot{\theta}}(t) = u_{\dot{\theta}}^{\text{opt}}(t) + u_{\dot{\theta}}^{\pi}(t)$, where $u_{\dot{\theta}}^{\pi}(t)$ is the pitch order given by the agent. Hence, when the agent acts on the pitch, it alters the trajectory of the predetermined standard scenario. This is to allow the agent to focus on learning failure cases without needing to learn a standard landing pattern.

4 Method and Implementation

The previous section outlined how the challenge of maximizing disagreement between two COM/MON units can be framed as a decision-making problem, opening the door to a myriad of potential algorithms for resolution. This section presents our chosen methodology. In this work, the state space which the agent can explore is technically infinite due to being continuous. We propose the use of neural networks to model policies as they are capable function approximators, even on continuous problems. We use the same neural architecture for all experiments: the neural network has 45 inputs (the state variables s_t), two fully-connected hidden layers of 64 neurons each with ReLU activation, and an output layer with 6 neurons, which encode the action a_t. Experiments on architectural parameters showed that deeper neural networks enhance performance but also increase computational cost. The chosen architecture presents an optimal balance between efficiency and computational resource requirements. We note the policy as $\pi_\phi : \mathcal{S} \mapsto \mathcal{A}$ and denote its parameters ϕ.

The objective we aim to maximize is the sum of reward from the environment cumulated by an agent along the trajectory :

$$\underset{\phi}{\texttt{argmax}} F(\phi) = \sum_{t=0}^{H} r_t = \sum_{t=0}^{H} \mathcal{R}(s_t, \pi_\phi(s_t)) \tag{1}$$

We propose to optimize the parameters of the network using three different methods. As a baseline, we use a Monte Carlo (MC) sampling method to generate policy networks which we refer to as "naive MC". We note that this baseline is commonly used in the literature of software testing. Specifically, we sample in λN networks $\phi_i \in \mathbb{R}^n$ uniformly in $[-10, 10]^n$, where λ is the number of sampled policy networks in a given epoch and N is the number of epochs.

Additionally, we study the use of Evolutionary Strategies (ES) [50] to optimize policies π. These methods have shown competitive results in policy search [52] and tend to drive the exploration of diverse policies. Specifically, we use a state of the art ES, CMA-ES [24] in its separable form [51]. These methods sample λ policies, also referred to as a population, from distribution $\mathcal{N}(\mu, \sigma)$ which is updated over iterations, or generations, in order to maximize an estimated gradient.

Finally, we propose to further encourage exploration of the state space by using the recent Curiosity-ES algorithm [31]. This algorithm combines Evolutionary Strategies with uncertainty bonuses, facilitating exploration in generic state spaces [5, 29, 47]. Specifically, Curiosity-ES uses the Intrinsic Curiosity Module (ICM) [47] to produce an intrinsic motivation. The ICM contains a neural network model which learns the dynamic of the environment and uses the error of the model's predictions as an intrinsic reward. For unseen areas, the error of the dynamic model are high, while in previously seen areas, these errors are expected to be low. This mechanism rewards agents for going in areas where the dynamics are not yet modelled accurately by the ICM.

In our implementation, the ICM is updated using samples of a replay buffer $\mathcal{D}$ which is filled with the trajectories of all policies ϕ_i sampled during evolution. For Curiosity-ES, the fitness is a weighted sum of the extrinsic and intrinsic fitness:

$$f_{\phi_i} = \eta f_{\phi_i}^i + (1 - \eta) f_{\phi_i}^e, \tag{2}$$

where the extrinsic fitness $f_{\phi_i}^e$ is the normalized sum of reward defined in Equation 1 and $\eta \in [0, 1]$ determines the importance of exploration. $f_{\phi_i}^e$ and $f_{\phi_i}^i$ are also normalized over the current generation, allowing for better control on the exploitation/exploration trade-off. Our neural network for the Curiosity Module (ICM) maintains a uniform architecture across experiments, featuring an encoder with three dense ReLU-activated layers of 64 and 32 neurons, ending in an 8-neuron linear output layer. The inverse model includes two ReLU-activated dense layers of 32 and 16 neurons,

branching into 'pitch' and 'shift' action layers with softmax activation. The forward model, with a 32-neuron ReLU layer and an 8-neuron linear layer, predicts environmental dynamics from the encoder's space.

Furthermore, we experimented with standard reinforcement learning baselines Proximal Policy Optimization (PPO) [53] and Soft Actor Critic (SAC)[23]. Despite hyperparameter tuning, these methods were unable to find high levels of COM/MON disagreement. As such, we exclude these methods from our analysis.

5 Experiments and Results

We first present the optimization processes of the three methods studied, showing that evolutionary strategies can find policies which create a high level of disagreement between the COM and MON units. We then evaluate the best policies from each method to understand how the agents create disagreement.

5.1 Policy Optimization

We evaluate the optimization of naive Monte Carlo sampling (naive MC), sCMA-ES, and Cusiosity-ES. All three methods are run for $N = 10000$ generations. We use the same population parameters for sCMA-ES and Curiosity-ES: $\sigma = 0.002$ and $\lambda = 50$. We performed optimization 5 times for each algorithm for statistical significance.

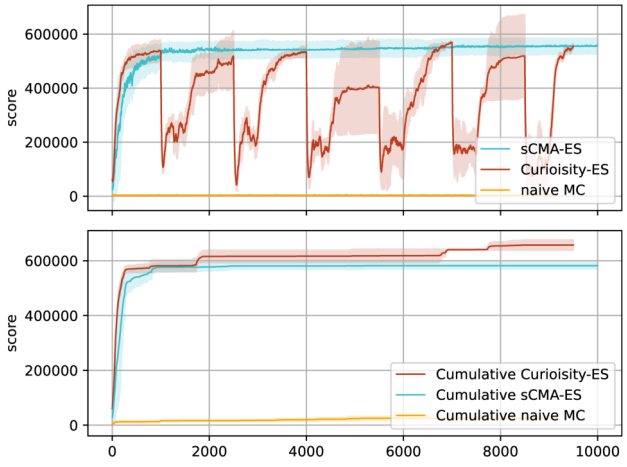

Figure 4: (top) Maximum fitness per generation during evolution for the three different methods. (bottom) Maximum fitness accumulated over evolution. Lines indicate the mean over 5 trials, and ribbons the standard deviation.

For Curiosity-ES, we split the search process between successive phases of hard exploration $\eta = 1$ and phases of extrinsic reward maximisation (reinforcement) $\eta = 0$. Each hard exploration phase lasts $N_e = 500$ generations, while the reinforcement phases last $N_r = 1000$. The population distribution of Curiosity-ES is randomly reinitialized at the end of each reinforcement phase, but the replay buffer and ICM are preserved. The ICM isn't restarted during evolution in order to keep track of what has been explored and to drive the exploration towards new scenarios.

In Figure 4, we note that the two ES methods are able to find highly rewarding policies which MC is unable to find. sCMA-ES is able to quickly find highly rewarding policies consistently over the five trials. While the evolution of Curiosity-ES has a higher standard deviation per generation than sCMA-ES, it outperforms sCMA-ES overall in cumulative maximum fitness, finding more rewarding policies during exploitation phases. The high variance in some phases of Curiosity-ES is due to the fact that the different trials might not focus on the exploration of the same area during the same phases, with some exploring areas of low reward. The oscillation of the phases of Curiosity-ES, between exploration and reinforcement, is seen in the top part of Figure 4 as the sharp drops of fitness when evolution is restarted.

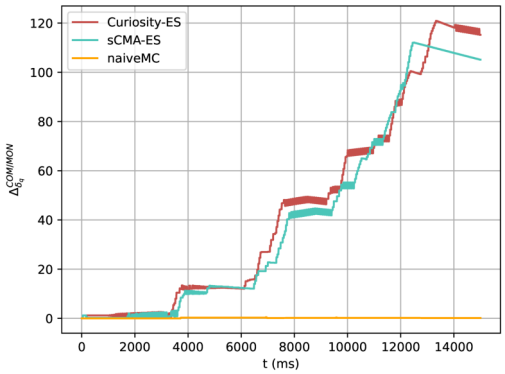

Figure 5: Temporal progression of the disagreement $\Delta\delta_q^{C/M}$ for the best policy found by each method

To better understand how the two methods control the system, we plot the progression of $\Delta\delta_q^{C/M}$ for the best policy found by the three methods. As can be seen in Figure 5, the disagreement between COM and MON is almost always larger for the Curiosity-ES policy. We also note that $\Delta\delta_q^{C/M}$ increases greatly for both ES policies at similar moments; we next present a detailed analysis of their behavior to understand how this disagreement is created.

5.2 Policy Evaluation

To understand the scenarios which generate high disagreement, we evaluate the best policy found by the different optimization methods while recording the states visited by each policy. We find that the policies use three main features principally to create disagreement: triggering bounces on landing, inflating the PID integrator, and the use of high-frequency signals which exploit the COM/MON phase shift.

5.2.1 Inducing Landing Bounces. As explained in subsection 3.3, we initialize the simulation in order to have the smoothest landing possible. However, we can see in Figure 6 that the best policies found by sCMA-ES and Curiosity-ES produce a landing with multiple bounces.

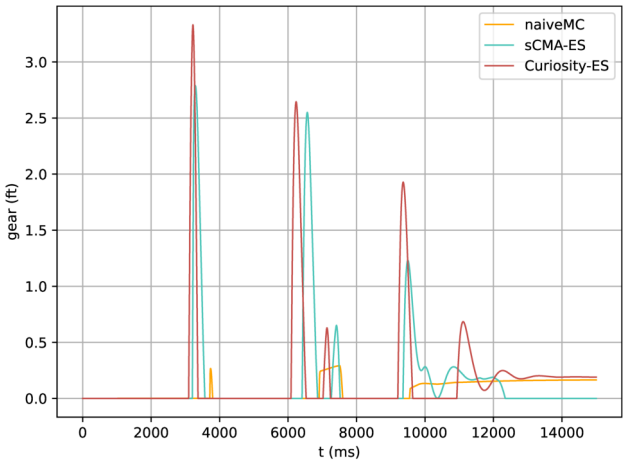

Figure 6: Temporal progression of the front gear compression for the three best policies. Both ES methods create many bounces to exploit the synchronization function.

This is an effective strategy for creating disagreement as, when b_C changes, the synchronizer $\mathsf{sync}(\delta_q^f, \delta_q^g, b_C)$ initiates a linear transition from one control law, ground or flight, to the other. Quick repeated bounces make this transition happen multiple times, and when COM and MON are out of phase, they do not start this transition at the same time. As such, the COM and MON may follow separate logic, one using the ground control law and the other the flight law, at different phases of synchronization.

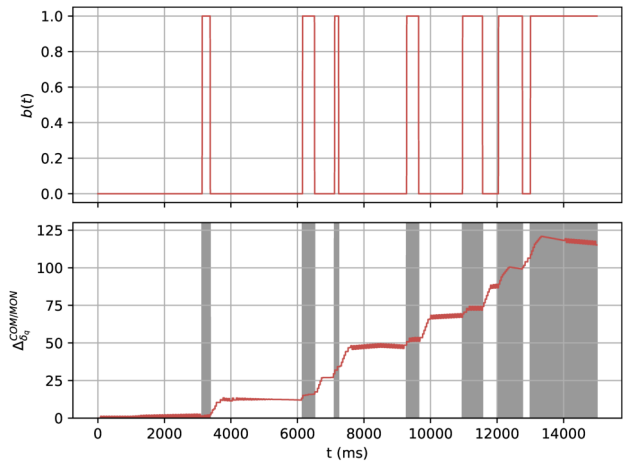

Figure 7: Temporal progression of the disagreement $\Delta_{\delta_q}^{C/M}$ (bottom) according to the impact b_C over time (top) following the best policy found by Curiosity-ES. Areas where b_C is true are highlighted in gray.

As we can see in Figure 7, following each change in b_C, the disagreement between COM and MON leaps, specifically after a short bounce. While there are other factors which contribute to the disagreement, the bounces have a direct influence on increasing $\Delta_{\delta_q}^{C/M}$.

Finally, it is interesting to note that the best policy found by naive MC does not have this feature, making small bounces similar to the original control scenario. This indicates that the bouncing behavior was not easy to find with random search. As this is a counter-intuitive behavior for a human pilot, we also consider that it demonstrates the ability of ES to find dangerous failure cases without human bias.

5.2.2 High-Frequency Signals. A second strategy which is indispensable for producing disagreement is the exploitation of the different sampling times between the two units to produce different inputs. While the two units follow the same logic, except in the previously seen case of repeated bounces, they can receive different inputs if there is a temporal shift between the units. As seen in Figure 2, each system updates its state variables by sampling, but when there is a temporal shift between the two units, they may sample different values of a variable. For example, every 10ms the COM samples $u_{\dot{\theta}}(t)$ and the MON samples $u_{\dot{\theta}}(t+\Delta_t^{C/M})$. Therefore, a good strategy is to create a temporal difference dynamic on $u_{\dot{\theta}}(t)$ so that the difference between two close samples is maximum.

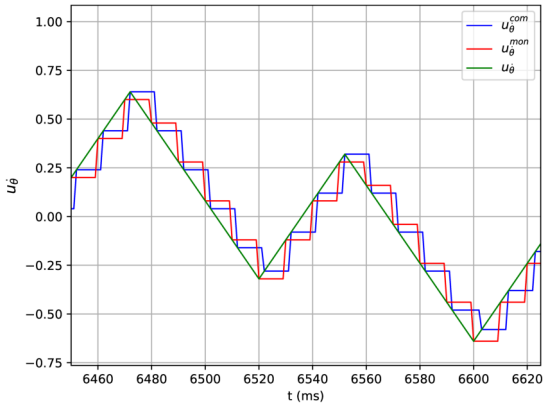

Figure 8: Temporal progression of $u_{\dot{\theta}}$, $u_{\dot{\theta}}^{COM}$ and $u_{\dot{\theta}}^{MON}$, following the best policy from Curiosity-ES

In Figure 8, we observe $u_{\dot{\theta}}(t)$ during a small time window of 150ms. We note that the agent varies the value of $u_{\dot{\theta}}(t)$ at an appropriate frequency so that the signal sample by COM (in blue) is always slightly above the signal sample from MON (in red). This strategy performed over the full trajectory of 15000ms largely contributes to the disagreement seen in Figure 5 as the control decisions are made using different state values.

It is also important to note the physical constraint on the change of $u_{\dot{\theta}}$ described in subsection 3.3, designed to be realistic. Agents optimized without these constraints produced a signal with greater oscillations that resulted in a much greater difference in state values. This highlights the exploration of behaviors that may seem unnatural to a human tester and approach the physical limits of the system.

5.2.3 Inflation of the PID Integrator. A final strategy for producing a high disagreement between COM and MON is to increase the values used to compute δ_q^C and δ_q^M. While these variables increase in both units, as COM and MON compute the order following the same logic, the magnitude of possible error $\delta_q^{C/M}$ increases with the magnitude of each unit's order. Specifically, when these variables are sampled at different times, the difference between the units increases as the magnitude of the state variables increases.

In a given unit, the variables that are most likely to diverge are the PID integrators. To inflate these variables, a policy can produce a pitch rate order that the system will not be able to respect. For example, a pitch rate order $u_{\dot{\theta}} < 0$ is physically impossible to respect when the front gear of the aircraft is compressed. As another example, in case of bounces with high amplitude (with the throttle in idle mode as is the case here), when the maximum amplitude of the bounce has been reached and the aircraft starts to come down, a pitch order rate $u_{\dot{\theta}} > 0$ is physically impossible.

The policies found by sCMA-ES and Curiosity-ES exploit this pattern. In Figure 9, we see the progression of $u_{\dot{\theta}}$ and b_C which indicates when the front gear is compressed. From $t = 5000ms$ to the end, the agent produces an average pitch order rate $u_{\dot{\theta}} < 0$ when $b_C = 1$ (front gear compressed) and $u_{\dot{\theta}} > 0$ when $b_C = 0$ (bounce phase). We postulate that the agent begins this feature at $t = 5000$ms, after the first bounce, as the control significantly affects the approach of the aircraft before the first bounce .However, after the first bounce, physically impossible controls are sometimes given in order to inflate the integrator values.

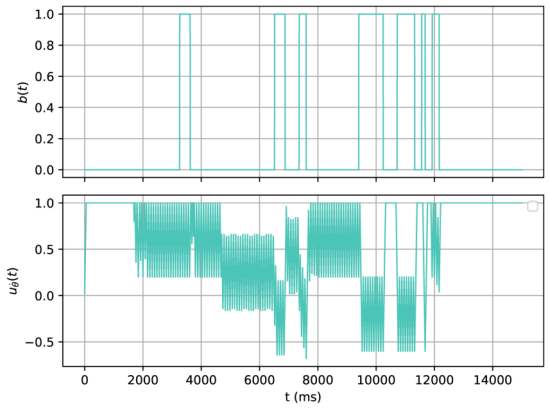

Figure 9: Temporal progression of $u_{\dot{\theta}}$ (bottom) with respect to the temporal evolution of the import boolean b (top), following the best policy from sCMA-ES

In Figure 9, we see the three main behaviors of the integrator: the bounces represented by the variable b, the high-frequency oscillations of the $u_{\dot{\theta}}$ variable which gives different state values to the two units, and the opposed pitch rate orders $u_{\dot{\theta}}$ that are physically impossible and increase the integrator values. We now study how the best policies found by the two methods differ in their use of these patterns.

5.3 Comparison of Methods

While the best policies found by sCMA-ES and Curiosity-ES exploit the same patterns to create disagreement, they rely on the different patterns to different degrees. As shown in Figure 6, both ES policies create similar bounces, only truly diverging at the end. However, their use of the sampling phase shift and integrator inflation patterns are different.

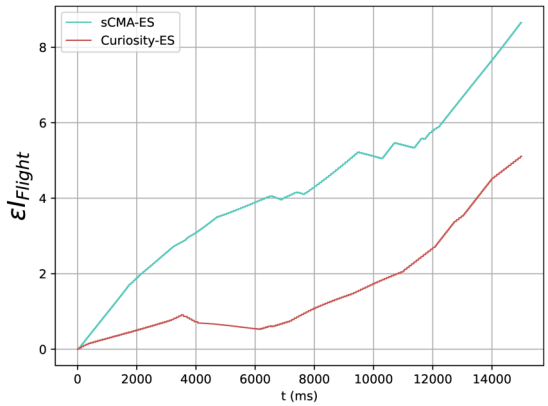

Figure 10: Temporal progression of integrated error ε_I contained in the PID of the flight law. sCMA-ES increases the integrator value more than Curiosity-ES.

In Figure 10, we show the evolution of the integrated error in the PID of the flight law $\varepsilon_I = K_i \int_0^t \varepsilon(x)dx$, where $K_i = 10$ for the flight law and 0 for the ground law. We note that the integrator grows significantly faster for the best policy found by sCMA-ES and is consistently larger than that of Curiosity-ES. Also, in Figure 6, we can see that the sCMA-ES policy is the only one for which the front gear is not compressed at the end of the simulation, resulting in a "wheeling". This is interesting, because when $b_C = 0$ (the case here), δ_q^f (flight law) is used to compute δ_q. Since $\delta_q^f > \delta_q^g$ because of the integrator, we can infer that the sCMA-ES policy has a tendency to focus principally on increasing the order of magnitude of δ_q so that when there is a disagreement, it is large.

However, when looking at the difference between the integrated error $\Delta\varepsilon_I^{C/M}$ in COM and MON, in Figure 11, we can see that the difference keeps growing for the Curiosity-ES policy and is significantly above that of the sCMA-ES policy. Hence, we can infer that the Curiosity-ES policy focuses on producing a signal that maximises the differences between the variables contained in the two units. While the value of the integrator is lower than that of sCMA-ES's best policy, the Curiosity-ES policy exploits the phase shift sampling pattern more, resulting in a larger difference in state variable values.

Analysing the best policies from the two methods permitsunderstanding the patterns used to create high disagreement. However, lower levels of COM/MON disagreement can still result in unit failure, and understanding the variety of cases which lead to failure is important for rigorous testing. We therefore evaluate the ability of the proposed optimization methods to find a diversity of failure cases.

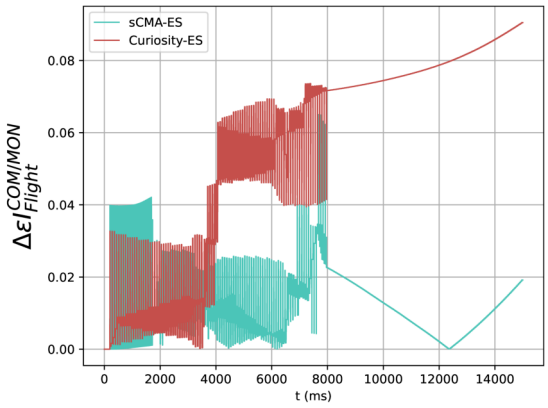

Figure 11: Temporal progression of the difference between the integrated error in the COM/MON flight law $\Delta\varepsilon_I^{C/M}$. Curiosity-ES maintains a higher difference by exploiting shifted sampling.

6 Diversity of Failure Cases

To understand the diversity of failure cases found by the ES methods, we evaluate all policies which surpass a fixed threshold of disagreement between the COM and MON. Specifically, we record all the policies for which the fitness is above $F_\tau = 450000$. We then analyze these policies in Figure 12, first analyzing the diversity of actions chosen by the policies and then the states visited.

6.1 Diversity of Chosen Actions

We first evaluate the diversity generated by the two methods by analyzing the $u_{\dot{\theta}}$ actions taken by the various policies. We use only the $u_{\dot{\theta}}$ actions as all policies showed similar behavior on the phase shift $\Delta_t^{C/M}$ action, which is to maximize the phase shift from the very beginning.

Analyzing the direct $u_{\dot{\theta}}$ over the horizon of $H = 15000$ms across multiple policies is intractable. Instead, we apply a Fourier transform to each signal and keep the 500 highest amplitude frequencies, describing each signal by these frequencies and their corresponding amplitudes (1000 total dimensions). We then use Principal Component Analysis (PCA) on the frequency representation of each policy to project it as a point in two dimensions, which we consider as the "policy space".

We can see in Figure 12 the result of this projection in the top three figures. The leftmost figure shows the projection of the policies found by sCMA-ES in green and the policies found during the hard exploration phases of Curiosity-ES in red. In the center figure, we add the policies found by Curiosity-ES during the reward maximisation phase in blue. In the right-most figure, we show policies from the three methods with their corresponding fitness value indicated in color.

As we would expect, the policies generated by Curiosity-ES cover much more of the policy space, especially during the Curiosity maximisation phase. When considering policies from the reinforcement phases, we see three clusters appear: one shared with sCMA-ES and two on the left which are only found by Curiosity-ES. Notably, one such cluster achieves maximum fitness. As the policy space focuses on the frequencies of $u_{\dot{\theta}}$, we interpret this group of policies as those similar to the Curiosity-ES policy in Figure 11, exploiting the sampling phase shift maximally with high-frequency oscillations.

6.2 Diversity of Visited States

Curiosity-ES is able to generate diversity in the policy space of chosen actions. However, in a constrained environment, two different policies may create the same sequences of visited states. Therefore, we also analyze the states sampled by all policies above the fitness threshold F_τ. For this comparison, we choose the $n = 20$ most rewarding states over the full episode as a representation of the trajectory. We flatten this n states vector and again use PCA to evaluate how the policies are distributed. We see in the bottom row of Figure 12 that Curiosity-ES visits a new part of the state space unexplored by sCMA-ES, represented by the cluster on the left of each figure. While these states were most likely found in an exploration phase, they lead to the highest reward found by either ES. This shows not only that Curiosity-ES can lead to discovering new failure states, but that exploration can also lead to regions of higher reward. This also demonstrates the advantage of exploration, as this group of failure states were only reached through explicit exploration. These reveal a different set of critical conditions for the control system.

7 Lessons Learned

Drawing from our findings, we now turn to the critical insights gained through our study. From an industrial perspective, our approach highlights a method that works very effectively in identifying these failure cases, while there was no automated solutions up to now on this specific system. This tool is particularly relevant as we were able to validate the key features employed by the agent to harm the system through discussions with industrial experts on COM/MON. In other words, the features identified by the agent are indeed the most relevant found to date on real systems. Naturally, these systems are designed to be robust against such challenges, but this underscores the credibility of the results obtained. Additionally, our latest study on identifying different clusters through PCA analysis was very enlightening for the engineers to whom the study was presented. This approach allows for a straightforward structuring of the post-analysis of results obtained by evolutionary methods. Identifying clusters using algorithms like K-means, followed by replaying the most effective cases based on the fitness criterion of each cluster, simplifies the evaluation process. From an implementation standpoint, our approach also highlights the robustness of evolutionary methods. In an industrial context, the complexity of testing methods may be a challenge due to the requirement of expertise, especially when using reinforcement learning for automated testing. In our study, we found that evolutionary approaches, which require simple management of relatively few hyperparameters, were easier to implement for an operational use-case than deep reinforcement learning approaches, which have more hyperparameters and require finer tuning. However, one drawback of evolutionary methods is that they are less sample-efficient than standard RL approaches and significant computational resources may be needed. In this study, we used a three-machine cluster,

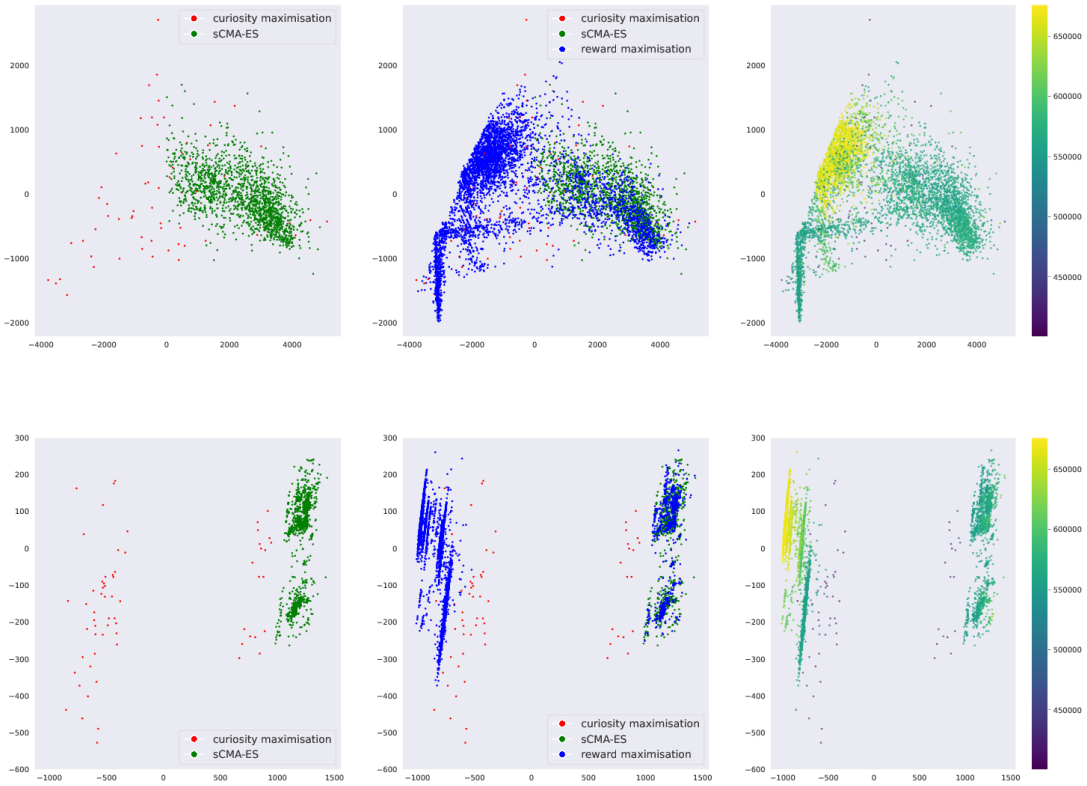

Figure 12: (top) PCA on frequency signal of actions. (bottom) PCA on $n = 20$ most rewarding states. (left) Test cases when Curiosity-ES maximizes curiosity. (center) Test cases including when Curiosity-ES also maximizes reward. (right) Fitness values of all identified test cases. In all figures, each dot represents the test cases sampled during the execution of each algorithm. Clusters illustrate the correlation between policies after the PCA transformation. Axes represent the dimensionless values of the first and second principal components.

each with an Intel Core i9, 64 GB RAM, and Red Hat Enterprise Linux, parallelized via the Ray library [45]. All experiments were conducted on CPUs, and there was no need for GPU utilization, and a computation time of 12 hours was required for five runs for each of the algorithms. In a continuation of this study, we are now focusing on replicating this study using an internal industrial simulator which is computationally more demanding. Our preliminary experiments on the more realistic industrial simulator indicate that to achieve the same results, a duration of one week (168 hours) will be necessary. This duration remains extremely reasonable given the criticality of the system and the time required for validating such a system.

8 Conclusion

In this article, we formulate the problem of testing a critical avionic system as a decision making problem. Specifically, we simulate a system based on a realistic aircraft flight control architecture and find possible failures in the COM/MON system based on subsystem disagreement. We show that ES can find a variety of policies which lead to critical unit failure and analyze the strategies of the best policies.

We propose the use of the recent Curiosity-ES [31] to enhance exploration and demonstrate its benefits in the context of test case coverage and explainability. We evaluate the diversity of actions and states in critical policies, showing that exploration can lead to more comprehensive testing and even more rewarding policies.

In this work, we used a fully simulated COM/MON unit and flight dynamics. While the short evaluation time in this simulation allowed the use of evolutionary strategies, the proposed methods can be computationally expensive and therefore may not be feasible for testing systems with high evaluation cost. In order to overcome this limitation, approaches that focus on improving the sample efficiency of ES methods with RL, such as [56], could allow for the use of these methods on other systems without incurring prohibitive computational costs. Preliminary testing with standard deep RL methods (specifically Soft Actor-Critic [23]) did not show sufficient exploration to surpass the naive MC baseline, but further study of

exploration in sample-efficient RL methods could aid in application to other systems.

While the proposed methods are able to find a diversity of failure cases, we note that these cases remain highly improbable. Indeed, results from the naive MC method show how uncommon a large disagreement between the COM and MON is. This further highlights the potential of Curiosity-ES to identify extremely unlikely events.

Finally, while our current focus is on aircraft control, the principles we lay out here have a wider applicability. They can be employed in various critical systems that require in-depth testing as long as a formulation as a sequential decision task is possible. The exhaustive exploration capabilities of Curiosity-ES could be especially useful in systems where a thorough investigation of potential failures is vital. Furthermore, the versatility of our approach enables its application across diverse systems. Practical applications could range from autonomous driving systems to control systems in industries like nuclear power, or even complex software systems. As with other RL methods, the key challenge is in framing the testing issue as a sequential decision-making problem.

Looking ahead, we plan to expand our research to apply these methods to additional critical avionic systems beyond the testing of Control/Monitoring disagreements. By doing so, we aim to demonstrate the broader applicability of our approach and contribute to improving safety precautions in a variety of system testing scenarios.

References

[1] 2022. ODI RESUME. https://static.nhtsa.gov/odi/inv/2022/INOA-PE22002-4385.PDF. Accessed: 23/03/2024.

[2] Yasasa Abeysirigoonawardena, Florian Shkurti, and Gregory Dudek. 2019. Generating adversarial driving scenarios in high-fidelity simulators. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 8271–8277.

[3] Andrea Arcuri. 2010. It does matter how you normalise the branch distance in search based software testing. In *2010 Third International Conference on Software Testing, Verification and Validation*. IEEE, 205–214.

[4] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. 2016. Test case prioritization of configurable cyber-physical systems with weight-based search algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 1053–1060.

[5] Arthur Aubret, Laetitia Matignon, and Salima Hassas. 2019. A survey on intrinsic motivation in reinforcement learning. *arXiv e-prints* (2019), arXiv–1908.

[6] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. 2020. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038* (2020).

[7] Richard Bellman. 1957. A Markovian Decision Process. *Indiana University Mathematics Journal* 6, 4 (1957), 679–684.

[8] Jon Berndt. 2004. JSBSim: An open source flight dynamics model in C++. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*. 4923.

[9] Thomas Bochot, Pierre Virelizier, Hélene Waeselynck, and Virginie Wiels. 2009. Model checking flight control systems: The Airbus experience. In *2009 31st International Conference on Software Engineering-Companion Volume*. IEEE, 18–27.

[10] Mohamed Boussaa, Olivier Barais, Gerson Sunyé, and Benoit Baudry. 2015. A novelty search approach for automatic test data generation. In *2015 IEEE/ACM 8th International Workshop on Search-Based Software Testing*. IEEE, 40–43.

[11] Oliver Bühler and Joachim Wegener. 2008. Evolutionary functional testing. *Computers & Operations Research* 35, 10 (2008), 3144–3160.

[12] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2018. Exploration by random network distillation. In *International Conference on Learning Representations*.

[13] Ying Chen, Shaowei Huang, Feng Liu, Zhisheng Wang, and Xinwei Sun. 2018. Evaluation of reinforcement learning-based false data injection attack to automatic voltage control. *IEEE Transactions on Smart Grid* 10, 2 (2018), 2158–2169.

[14] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. 2018. Back to basics: benchmarking canonical evolution strategies for playing Atari. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 1419–1426.

[15] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. 2018. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *Advances in neural information processing systems* 31 (2018).

[16] Antoine Cully. [n. d.]. Autonomous Skill Discovery with Quality-Diversity and Unsupervised Descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2019-07-13) *(GECCO '19)*. Association for Computing Machinery, 81–89. https://doi.org/10.1145/3321707.3321804

[17] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. [n. d.]. Robots That Can Adapt like Animals. 521, 7553 ([n. d.]), 503–507. Issue 7553. https://doi.org/10.1038/nature14422

[18] Minghui Dai, Zhou Su, Qichao Xu, and Weiwei Chen. 2019. A Q-learning based scheme to securely cache content in edge-enabled heterogeneous networks. *IEEE Access* 7 (2019), 163898–163911.

[19] Didier Essame, Jean Arlat, and David Powell. 1999. Padre: A protocol for asymmetric duplex redundancy. In *Dependable Computing for Critical Applications 7*. IEEE, 229–248.

[20] Shuo Feng, Haowei Sun, Xintao Yan, Haojie Zhu, Zhengxia Zou, Shengyin Shen, and Henry X Liu. 2023. Dense reinforcement learning for safety validation of autonomous vehicles. *Nature* 615, 7953 (2023), 620–627.

[21] Federico Formica, Nicholas Petrunti, Lucas Bruck, Vera Pantelic, Mark Lawford, and Claudio Menghi. 2023. Test case generation for drivability requirements of an automotive cruise controller: An experience with an industrial simulator. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1949–1960.

[22] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 318–328.

[23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 1861–1870.

[24] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9, 2 (2001), 159–195.

[25] Junda He, Zhou Yang, Jieke Shi, Chengran Yang, Kisub Kim, Bowen Xu, Xin Zhou, and David Lo. 2024. Curiosity-Driven Testing for Sequential Decision-Making Process. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 949–949.

[26] Gunel Jahangirova, Andrea Stocco, and Paolo Tonella. 2021. Quality metrics and oracles for autonomous vehicles testing. In *2021 14th IEEE conference on software testing, verification and validation (ICST)*. IEEE, 194–204.

[27] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J Kochenderfer. 2018. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1–7.

[28] Mark Koren and Mykel J Kochenderfer. 2019. Efficient autonomy validation in simulation with adaptive stress testing. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 4178–4183.

[29] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems* 29 (2016).

[30] Alexandr Kuznetsov, Yehor Yeromin, Oleksiy Shapoval, Kyrylo Chernov, Mariia Popova, and Kostyantyn Serdukov. 2019. Automated software vulnerability testing using deep learning methods. In *2019 IEEE 2nd Ukraine Conference on Electrical and Computer Engineering (UKRCON)*. IEEE, 837–841.

[31] Paul-Antoine Le Tolguenec, Emmanuel Rachelson, Yann Besse, and Dennis G. Wilson. 2023. Curiosity Creates Diversity in Policy Search. *ACM Trans. Evol. Learn. Optim.* (jun 2023). https://doi.org/10.1145/3605782

[32] Ritchie Lee, Mykel J Kochenderfer, Ole J Mengshoel, Guillaume P Brat, and Michael P Owen. 2015. Adaptive stress testing of airborne collision avoidance systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*. IEEE, 6C2–1.

[33] Joel Lehman and Kenneth O Stanley. [n. d.]. Exploiting Open-Endedness to Solve Problems through the Search for Novelty.. In *ALIFE* (2008). 329–336.

[34] Joel Lehman and Kenneth O Stanley. 2011. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation* 19, 2 (2011), 189–223.

[35] Joel Lehman and Kenneth O Stanley. 2011. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 211–218.

[36] Chao Li, Wen Zhou, Kai Yu, Liseng Fan, and Junjuan Xia. 2019. Enhanced secure transmission against intelligent attacks. *IEEE Access* 7 (2019), 53596–53602.

[37] Bing Liu, Shiva Nejati, Lucia, and Lionel C Briand. 2019. Effective fault localization of automotive Simulink models: achieving the trade-off between test oracle effort and fault localization accuracy. *Empirical Software Engineering* 24 (2019), 444–490.

[38] Xing Liu, Hansong Xu, Weixian Liao, and Wei Yu. 2019. Reinforcement learning for cyber-physical systems. In *2019 IEEE International Conference on Industrial Internet (ICII)*. IEEE, 318–327.

[39] Zengguang Liu, Xiaochun Yin, and Yuemei Hu. 2020. CPSS LR-DDoS detection and defense in edge computing utilizing DCNN Q-learning. *IEEE Access* 8 (2020), 42120–42130.

[40] Reza Matinnejad, Shiva Nejati, and Lionel C Briand. 2017. Automated testing of hybrid simulink/stateflow controllers: industrial case studies. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 938–943.

[41] Reza Matinnejad, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. 2016. Automated test suite generation for time-continuous simulink models. In *proceedings of the 38th International Conference on Software Engineering*. 595–606.

[42] Phil McMinn. 2011. Search-based software testing: Past, present and future. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 153–163.

[43] Paulina Stevia Nouwou Mindom, Amin Nikanjam, and Foutse Khomh. 2022. A comparison of reinforcement learning frameworks for software testing tasks. *arXiv preprint arXiv:2208.12136* (2022).

[44] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[45] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 561–577.

[46] Justin Norden, Matthew O'Kelly, and Aman Sinha. 2019. Efficient Black-box Assessment of Autonomous Vehicle Safety. *arXiv e-prints* (2019), arXiv–1912.

[47] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*. PMLR, 2778–2787.

[48] Peter Puschner and Roman Nossal. 1998. Testing the results of static worst-case execution-time analysis. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*. IEEE, 134–143.

[49] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[50] Ingo Rechenberg. 1978. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*. Springer, 83–114.

[51] Raymond Ros and Nikolaus Hansen. 2008. A Simple Modification in CMA-ES Achieving Linear Time and Space Complexity. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature—PPSN X-Volume 5199*. 296–305.

[52] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv e-prints* (2017), arXiv–1703.

[53] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[54] Andrea Stocco, Brian Pulfer, and Paolo Tonella. 2022. Mind the gap! a study on the transferability of virtual vs physical-world testing of autonomous driving systems. *IEEE Transactions on Software Engineering* (2022).

[55] Andrea Stocco, Brian Pulfer, and Paolo Tonella. 2023. Model vs system level testing of autonomous driving systems: a replication and extension study. *Empirical Software Engineering* 28, 3 (2023), 73.

[56] Jörg Stork, Martin Zaefferer, Nils Eisler, Patrick Tichelmann, Thomas Bartz-Beielstein, and AE Eiben. 2021. Behavior-based neuroevolutionary training in reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1753–1761.

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

[58] Nigel Tracey, John Clark, Keith Mander, and John McDermid. 1998. An automated framework for structural test-data generation. In *Proceedings 13th IEEE International Conference on Automated Software Engineering (Cat. No. 98EX239)*. IEEE, 285–288.

[59] Pascal Traverse, Isabelle Lacaze, and Jean Souyris. 2004. Airbus fly-by-wire: A total approach to dependability. In *Building the Information Society: IFIP 18th World Computer Congress Topical Sessions 22–27 August 2004 Toulouse, France*. Springer, 191–212.

[60] Joachim Wegener, André Baresel, and Harmen Sthamer. 2001. Evolutionary test environment for automatic structural testing. *Information and software technology* 43, 14 (2001), 841–854.

[61] Joachim Wegener and Matthias Grochtmann. 1998. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems* 15 (1998), 275–298.

[62] Joachim Wegener and Frank Mueller. 2001. A comparison of static analysis and evolutionary testing for the verification of timing constraints. *Real-time systems* 21 (2001), 241–268.

[63] Joachim Wegener, Harmen Sthamer, Bryan F Jones, and David E Eyres. 1997. Testing real-time systems using genetic algorithms. *Software Quality Journal* 6 (1997), 127–135.

[64] Junhui Zhang and Jitao Sun. 2019. A game theoretic approach to multi-channel transmission scheduling for multiple linear systems under DoS attacks. *Systems & Control Letters* 133 (2019), 104546.

[65] Tao Zhang, Xiaohui Kuang, Zan Zhou, Hongquan Gao, and Changqiao Xu. 2019. An intelligent route mutation mechanism against mixed attack based on security awareness. In *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.

[66] Yan Zheng, Yi Liu, Xiaofei Xie, Yepang Liu, Lei Ma, Jianye Hao, and Yang Liu. 2021. Automatic web testing using curiosity-driven reinforcement learning. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 423–435.

Received 2024-04-12; accepted 2024-07-03