Deep-RL-based Nonlinear Adaptive Flight Control: on the gap between simulation and reality

Adrian Beňo
Delft University of Technology, Kluyverweg 1, 2629HS Delft, The Netherlands
M.Sc. Student, Faculty of Aerospace Engineering, A.Beno@student.tudelft.nl

This paper presents a novel corrective algorithm bridging the gap between simulation and reality by online fine-tuning an offline pre-trained deep reinforcement learning agent. The novel control architecture is inspired by the incremental model-based heuristic dynamic programming, which is described together with the basics of reinforcement learning first. This novel control architecture is applied in an illustrative control environment. It was found that the corrective algorithm can help reach the desired reference state in an environment governed by moderately different dynamics from those used during pre-training of the reinforcement learning agent.

I. Introduction

The need for adaptive robust flight controllers has become apparent as the systems which must be controlled have substantially grown in complexity. One could continue trying to design ever more complex architectures of, for example, cascaded PID control laws, but this approach is inefficient for highly nonlinear systems. Another problem faced by modern controllers is the lack of knowledge of the plant dynamics. Although high-fidelity system models could be developed to better approximate the true system dynamics, these attempts usually cannot keep the pace with complex aerodynamic phenomena experienced by modern aircraft, such as fighters or drones, not to mention their imperative unavailability before the first flight. Lastly, it cannot always be assumed that the system dynamics remain constant during flight. Adaptability is one of the crucial properties of future flight controllers, which will help cross the bridge between simulation and reality.

This research is inspired by transitioning UAVs, such as given in Figure 1, because of their advantageous flight performance. Unfortunately, such systems are high-dimensional, nonlinear and unstable. One of the recent approaches to tackle the problem of high-dimensionality and nonlinearity is *deep reinforcement learning* (DRL) [1]. This approach alleviates the controller complexity from its designer, though at the price of a black-box controller.

DRL has already been successfully used in various control architectures, such as online tuning of PID gains [2] or direct control [3], [4]. Conventionally, DRL requires extensive training phase to learn the control laws. Sufficient online training time is usually not available for unstable systems, hence it must be trained offline, or as an observer of another controller which guarantees safe online training [5]. However, offline training requires good knowledge of system dynamics, which is usually not known.

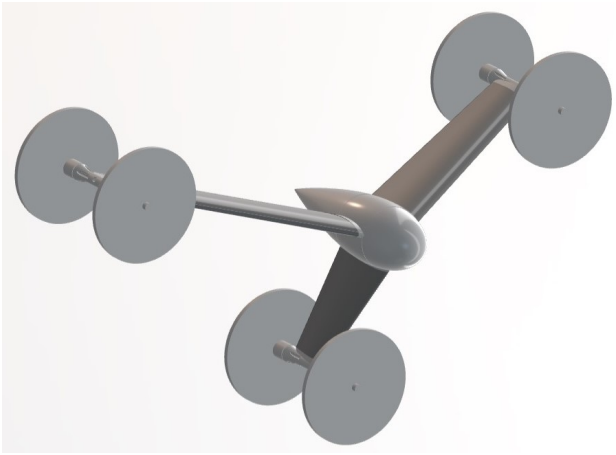


Figure 1. Transitioning UAV, an example of a nonlinear unstable system whose dynamics are not completely known due to complex aerodynamic phenomena and manufacturing inaccuracies.

One of the approaches to tackle the problem of adaptability is *incremental model-based heuristic dynamic programming* (IHDP), [6]. DRL agents can benefit from IHDP, yielding an adaptive combined control law; however, it still has limitations in handling highly unstable systems. This is because of the time needed for learning. Consequently, even IHDP-based adaptive control cannot be readily deployed on drone-like systems.

The tools to address both high controller complexity and adaptability are thus available separately. Combination of traditional DRL with IHDP is proposed in this paper to bridge the gap between simulation and reality, resulting in a novel corrective algorithm which can online safely fine-tune crude offline-learned control laws. Though, it is only implemented on a system representative of a true drone system, as a proof of concept.

The rest of the paper is structured as follows. The basics of DRL and IHDP are explained in section II. The shortcomings of DRL and IHDP in the context of their deployment outside of training environments and the proposed corrective algorithm are discussed in section III. Results are presented in section IV. The concluding section discusses the advantages and disadvantages of the corrective algorithm, as well as challenges and future research directions.

II. Background

A. Deep Reinforcement Learning

Reinforcement learning as a sub-field of machine learning is the "computational approach to learning from interaction" [1]. The learning phase consists of collecting reward R_{t+1} after taking action $\vec{a}_t$, given state $\vec{s}_t$. The environment then propagates the agent to the next state $\vec{s}_{t+1}$, at which point these steps begin to repeat in a loop. This is schematized in Figure 2.

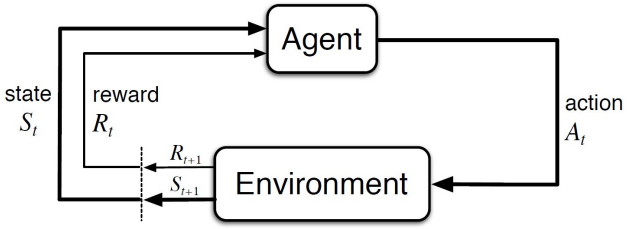


Figure 2. RL agent learning scheme. From [1].

The agent tries to learn a policy π, a mapping from state $\vec{s} \in S$ to action $\vec{a} \in A$, which maximizes the future cumulative collected reward $G_t = \sum_{t=t_0}^{T} R_t$. The learned policy can be approximated by any function approximator, such as a tabular method or an *artificial neural network* (ANN). In this case, the RL agent is referred to as *deep* RL agent. The environment in which the agent seeks to learn its optimal policy π^* should satisfy the Markov property[1]. In case it does not, convergence to the optimal policy cannot be guaranteed, but learning of a good policy is still possible. Hence, an RL agent can be mathematically formalized as living in a *Markov decision process* (MDP), trying to learn the policy resulting in the highest collected reward.

To ease the reading of the following sections we now briefly introduce one of the most widely used RL schemes: *actor-critic* (AC) policy gradient methods. *Actor* refers to the action-taking part of an AC agent. It is responsible for state-space exploration, learning of the optimal policy and storing it. In case of episodic environments, the performance measure J of a policy π parameterized by $\vec{w}_a$ is given as the value v of the initial state $\vec{s}_0$, where v denotes the expected cumulative reward to be collected by the end of the episode. This is formally written in Equation 1.

$$J(\vec{w}_a) \equiv v_{\pi_{\vec{w}_a}}(\vec{s}_0) \tag{1}$$

Actor updates its policy so that J increases. This corresponds to updates in the direction of the gradient $\nabla_{\vec{w}_a} J(\vec{w}_a)$. This gradient can be found using the policy gradient theorem [1]. The update rule is given in Equation 2,

$$\vec{w}_a \leftarrow \vec{w}_a + \alpha_a \nabla_{\vec{w}_a} J(\vec{w}_a) \tag{2}$$

where α_a denotes the actor's step size.

Critic refers to the value-approximating part of an AC agent. It provides the actor with information on how valuable a state is. This allows the agent to progressively select actions which lead to states with higher value. Critic can also be expressed by any function approximator, such as an ANN, and its updates are to minimize the difference between the critic's approximated value $\tilde{v}$ and the true value v^* of

[1] Traditional RL agents intrinsically assume the environment to be Markovian.

any state. The letter is, of course, not known and thus bootstrapped. This is formally written in Equation 3,

$$\vec{w}_c \leftarrow \vec{w}_c - \alpha_c \nabla_{\vec{w}_c} \frac{1}{2}(v^* - \tilde{v})^2$$

$$\vec{w}_c \leftarrow \vec{w}_c + \alpha_c (v^* - \tilde{v}) \nabla_{\vec{w}_c} \tilde{v} \tag{3}$$

where α_c denotes the critic's step size.

Most AC agents make use of the previous architecture and differ, for example, in their approach to state-space exploration, stability of updates or bootstrapping method. Examples of most popular AC agents include SAC [7], DDPG [8] and PPO [9].

B. Incremental Model-Based Heuristic Dynamic Programming

IHDP [6] is a model-free adaptive control extension of a typical AC controller. IHDP enhances AC agent's architecture presented in subsection II.A with an online-identified incremental plant model. This linear time-varying approximation of the true nonlinear system dynamics is used to inform the actor about the future consequences of its current action. The actor can then better adjust its parameterization at each time step, as system dynamics changes. IHDP control architecture is shown in Figure 3.

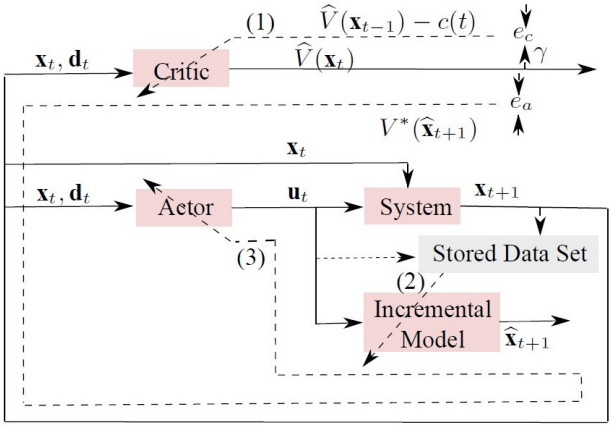


Figure 3. IHDP architecture. From [6].

The incremental model is a least-squares fit of the past M state-action ($\vec{x}$-$\vec{u}$) transitions, used to compute the system transition matrix F and the control effectiveness matrix G in Equation 4.

$$\Delta\vec{x}_{t+1} \simeq F_{t-1}\Delta\vec{x}_t + G_{t-1}\Delta\vec{u}_t \tag{4}$$

It can be used to inform the policy update because such plant model is differentiable. The actor updates the policy in the spirit of a traditional AC framework to increase the value of the next state. The corresponding update rule is given in Equation 5.

In this formulation, the value function is defined in terms of a cost-to-go objective, where rewards are typically negative and the goal is to minimize cumulative cost. Consequently, maximizing the value function is equivalent to minimizing the cost-to-go function. The gradient-based update in Equation 5 therefore proceeds in the direction of decreasing cost, i.e., along the negative gradient. This encourages the agent to reach the reference state as quickly as possible while also allowing additional penalty terms in the rewards R_t to discourage undesired behaviour, such as exceeding critical drone attitude.

$$E_a(t) = \frac{1}{2}\tilde{v}^2(\vec{x}_t)$$

$$\vec{w}_a \leftarrow \vec{w}_a - \alpha_a \frac{\partial E_a(t+1)}{\partial \vec{w}_a(t)} \tag{5}$$

$$\frac{\partial E_a(t+1)}{\partial \vec{w}_a(t)} = \frac{\partial E_a(t+1)}{\partial \tilde{v}(\hat{x}_{t+1})} \frac{\partial \tilde{v}(\hat{x}_{t+1})}{\partial \hat{x}_{t+1}} \frac{\partial \hat{x}_{t+1}}{\partial \vec{u}_t} \frac{\partial \vec{u}_t}{\partial \vec{w}_a(t)}$$

Here $\frac{\partial \hat{x}_{t+1}}{\partial \vec{u}_t} = G_{t-1}$ is given by the differentiable incremental model. $\frac{\partial \vec{u}_t}{\partial \vec{w}_a(t)}$ and $\frac{\partial \tilde{v}(\hat{x}_{t+1})}{\partial \hat{x}_{t+1}}$ are standard derivatives taken across the actor and critic ANNs (or any other differentiable function approximators). Critic updates itself in the standard AC manner, given in Equation 3.

III. Methodology

This section firstly justifies the need for the proposed corrective algorithm in subsection III.A. The architecture of this corrective controller is described in subsection III.B. The results obtained in a drone simulation environment are shown and discussed in subsection III.C. The obtained results reveal that the drone simulation environment cannot be used to showcase the capabilities of the proposed corrective algorithm at the time of writing. Hence, the Gym pendulum environment will be used instead in the rest of this paper to present the corrective algorithm as a proof-of-concept. A brief overview of the Gym pendulum environment is given in subsection III.D.

A. The Need for a Novel Corrective Algorithm

The necessity of the corrective algorithm stems from the inherent shortcomings of DRL and IHDP. On the one hand, offline trained agents can be made very robust through exhaustive state-space exploration and essentially unlimited number of training steps. On the other hand, offline training requires knowledge of the system dynamics, which is usually not available or is low-fidelity. A DRL agent outside of simulation must then be corrected during flight to account for the real-world aerodynamic phenomena or manufacturing deficiencies.

What happens when this is not the case is demonstrated with a DRL agent trained to stabilize itself to hover at $z = 1$ [m] and $y = 0$ [m]. The agent was trained in a custom-built drone environment, which adheres to the Gym API [10], using Stable-Baselines3 PPO implementation [11], mimicking the research done in Ref. [12] in every other aspect. It was then transferred to the "real-world" environment with higher drone weight or small constant force acting on it. This is to mimic the manufacturing deficiencies or complex aerodynamic phenomena, unknown during the training. The dynamical model of the drone follows [13]. This simple model captures all but aerodynamic phenomena. The results are shown in Figure 4.

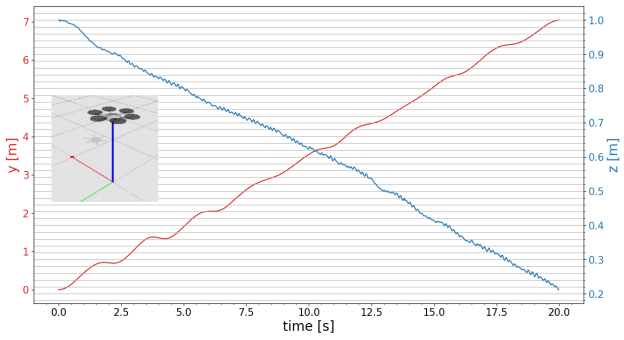


Figure 4. Response of an offline-trained PPO agent to a higher weight (blue) and a small constant external force (red), mimicking the unknown discrepancies between the training environment and real world conditions. The agent was trained to reach hover state at $(z, y) = (1, 0)$ [m]. The non-adaptive agent initially attempts to reach a zero-velocity state; however, its performance deteriorates over time, completely missing the reference state.

It is evident that the drone cannot adequately mitigate real-world disturbances or adapt to dynamics that were not present in the training environment. Ref. [6] shows that integrating DRL with IHDP leads to promising results in fault-tolerant adaptive control by enabling online fine-tuning of a DRL agent. However, according to the present author, this approach also exhibits hidden flaws that are not explored or addressed in that work. Two fundamental limitations are outlined below.

- The pre-trained value function $\tilde{v}$ is only a good approximation of the true value function v^* for states that were sufficiently frequently visited during training. In reinforcement learning, it is common that certain regions of the state space are visited only rarely, for example due to convergence to a local optimum from which the agent does not escape. Consequently, the mismatch between the real-world dynamics and those assumed in the training environment may cause the aircraft to encounter precisely these rarely visited states, in which $\tilde{v}$ is poorly defined or effectively arbitrary. As a result, the controller's response and subsequent online learning become unreliable. This phenomenon was observed to lead to degradation of previously learned policies.

- There is no natural termination of learning based solely on value-function ascent, since it is always possible to further improve the policy in a way that increases the estimated state value. Consequently, an artificial stopping criterion must be imposed by the designer in order to prevent overfitting to local regions near the target reference states at the expense of global performance. This is a well-known issue in machine learning.

B. The Architecture of the Corrective Algorithm

A novel corrective algorithm is now proposed, which attempts to online robustly fine-tune an offline-trained RL agent, under the assumption that the discrepancy between the simulated training environment and the real-world dynamics does not require the agent to escape any local optimum, but merely move towards its shifted center. In other words, there must be no physical constraints in the real world that prevent the

agent from following the same trajectory as in the training environment, but only require adjustments in its control actions.

This corrective algorithm is based on a *target* actor. Target networks, such as target actors and target critics, are usually introduced to stabilize the training by avoiding the situation of "chasing your own tail" in the updates of critics. This is best intuitively explained in [14]. Instead of bootstrapping v^* in Equation 3 by the critic itself, v^* is bootstrapped from a target critic, which is updated less frequently. An example of an RL agent utilizing target networks is DDPG [8].

Consider now the extreme case in which a target *actor* network is never updated after training. In this case, the target actor must encode knowledge of the ground truth, i.e., what is achievable not only in simulation but also in the real world, thereby implicitly relying on the underlying assumption.

Building on this idea, the corrective algorithm consists of two networks: *learning actor* (L) and *target actor* (T). Initially, both are the same $\vec{w}_L = \vec{w}_T$ and correspond to the policy learned during the offline training. The learning actor then updates its policy online so that the expected outcome $\hat{x}_{t+1}$ of its action $\vec{u}_t$ in the real world approaches the outcome $\bar{x}_{t+1}$ of the target actor's action $\bar{u}_t$ in the training environment. Formally written, the corrective algorithm aims to minimize the L^2 norm e_a of the difference between $\hat{x}_{t+1}$ and $\bar{x}_{t+1}$.

$$e_a = \|\hat{x}_{t+1} - \bar{x}_{t+1}\|_2$$

$$E_L = \frac{1}{2} e_a^2$$

$$\vec{w}_L \leftarrow \vec{w}_L - \alpha_L \frac{\partial E_L(t+1)}{\partial \vec{w}_L(t)} \tag{6}$$

$$\frac{\partial E_L(t+1)}{\partial \vec{w}_L(t)} = \frac{\partial E_L(t+1)}{\partial e_a(t+1)} \frac{\partial e_a(t+1)}{\partial \hat{x}_{t+1}} \frac{\partial \hat{x}_{t+1}}{\partial \vec{u}_t} \frac{\partial \vec{u}_t}{\partial \vec{w}_L(t)}$$

This is possible to do online due to the IHDP-inspired update rule, which informs the learning actor about the consequences of its actions via $\hat{x}_{t+1}$, supplied by the incremental model. The architecture of this corrective algorithm is given in Figure 5. The update rule of the learning actor is given in Equation 6.

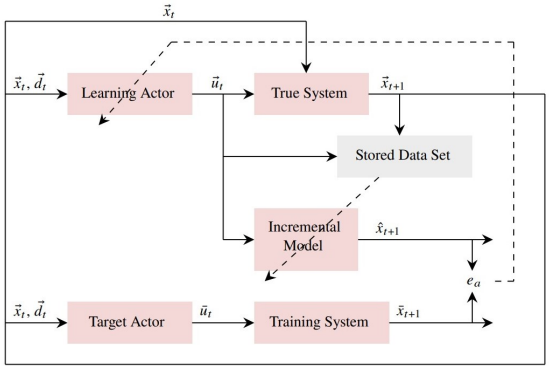


Figure 5. Corrective algorithm architecture. The training environment (system) provides the target states $\bar{x}_{t+1}$ that the target actor aims to reach from the current real-world state $\vec{x}_t$. An incremental model predicts the next state $\hat{x}_{t+1}$ resulting from the learning actor's action in the real system. The learning actor is then updated such that its predicted next state $\hat{x}_{t+1}$ approaches the target state $\bar{x}_{t+1}$ generated by the target actor in pursuit of the reference state $\vec{d}_t$.

C. Drone Environment

As already mentioned, it was not possible to demonstrate the deployment of the corrective algorithm in the drone environment at the time of writing of this paper. Importantly, this limitation does not originate from the corrective algorithm itself, which was found to operate as intended. Instead, the issue arises from inaccurate state predictions produced by the incremental model in the drone environment. These inaccuracies are caused by ill-posed least-squares fits, resulting in excessively large condition numbers due to insufficient persistent excitation in the collected data. This section discusses the possible causes of these issues and potential approaches to mitigate them. Despite considerable effort, the condition numbers could not be reduced to acceptable levels, preventing the drone environment from being used as a test bed for the corrective algorithm.

The first reason for large condition numbers is a high-frequency action signal, which is an artifact of the learning process. Simulation environment can instantaneously respond to any action signal, however, a physical engine cannot. Fortunately, this can be mitigated to high extent with *regularization* of ANNs[2]

[2]Filtering of the action signal should be avoided as this

[15]. Instead of adding a frequency related loss L_{reg} inside the reward function R_t, this loss can be directly fed into the updates of the ANN. This corresponds to adding the loss term to the performance measure in Equation 1, formally written below,

$$J(\vec{w}_a) \equiv v_{\pi_{\vec{w}a}}(\vec{s}_0) - \alpha_{reg} L_{reg}$$

where α_{reg} is a parameter used to control the comparative dominance of L_{reg}. This approach is applied on RL problems in Ref. [16] and named *conditioning for action policy smoothness* (CAPS). Bypassing the agent in this regard is highly beneficial. Otherwise the agent would have to disentangle the frequency related loss from an already complicated reward function R_t. The effect of CAPS can be seen in Figure 6. Although, meticulous attention should be paid to the weight α_{reg} of the regularization term, as this should not become more dominant than the RL performance term v_π itself.

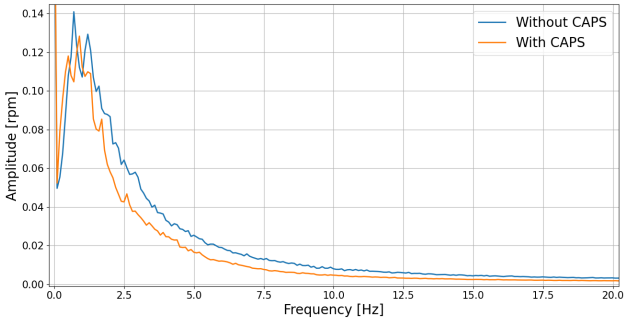


Figure 6. Average amplitude spectrum of the normalized action signal for a SAC agent trained with and without CAPS. Both agents were trained under identical conditions, differing only in the use of CAPS, with $\alpha_{reg} = 0.5$. The frequency analysis confirms that CAPS effectively suppresses high-frequency signal components.

The second cause of large condition numbers is the similarity between consecutive actions or states. Controllers commonly generate saturated control inputs over multiple consecutive time steps, repeatedly producing the same maximum or minimum action signal. However, even action signals with only minor variation may be sufficient to render an incremental model ill-conditioned. This results from insufficient system excitation, which causes information about the system dynamics to vanish.

A similar issue arises in the state space. While some state variables evolve over time, others may remain nearly constant. Such insufficient state variation can likewise render the incremental model ill-conditioned.

Several solutions besides CAPS are proposed to reduce the condition numbers. One approach is to track a time-varying reference signal, for example a sinusoidal reference, which empirical observations suggest provides sufficient excitation. More generally, persistent excitation of the system and an appropriate choice of excitation signal are crucial for the proper functioning of an incremental model, as analysed in Ref. [17]. Similarly, an appropriate selection of state variables can also reduce the condition numbers. In particular, the set of state variables used by the incremental model may be adjusted dynamically to include only variables exhibiting sufficient variation.

Furthermore, a mechanism must be implemented to halt model updates when condition numbers become unacceptable. This does not necessarily imply that the model ceases to represent the true system dynamics at those times. In fact, the underlying system dynamics often remain unchanged during periods of equilibrium.

Despite the aforementioned measures, the condition numbers in simulation could not be reduced to acceptable levels. Therefore, a substitute environment is introduced to demonstrate the corrective algorithm as a proof of concept.

D. Pendulum Environment

The Gym pendulum environment [10] was chosen as a classical control benchmark for testing the corrective algorithm. This choice is motivated by its ability to capture key properties of the drone environment, such as an unstable equilibrium at the reference state (the upright pendulum position), albeit at a smaller scale. A snapshot of the environment is shown in Figure 7 for illustrative purposes

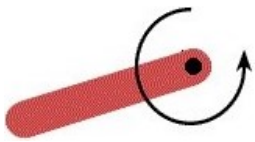


Figure 7. Snapshot of the Gym pendulum environment. The arrow symbolizes the applied torque.

introduces phase shift in the filtered signal. This phase shift could be fatal for unstable systems.

The goal in this environment is to swing up the pendulum and stabilize it in the upright position by applying a control torque T_c. However, the maximum available torque is insufficient to directly swing the pendulum from the downward equilibrium. Consequently, the agent must first learn to generate oscillatory motion in order to accumulate sufficient momentum to overcome gravity and reach the upright state.

The state vector is three-dimensional (as opposed to 13, as suggested in [12] for full drone dynamics) and consists of the pendulum angular velocity ω (with positive values corresponding to clockwise rotation) and the sine and cosine of the pendulum angle θ, where $\theta = 0$ corresponds to the upright position. This representation is analogous to [3], where all nine elements of the direction cosine matrix are used to represent the drone's attitude in order to avoid singularities. The nonlinearity of the pendulum dynamics arises from the sine term, as shown in Equation 7.

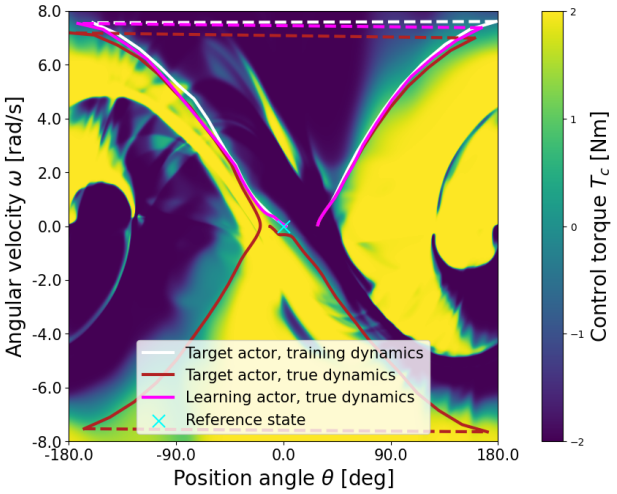
$$\dot{\omega} = \frac{3g}{2l} \sin\theta + \frac{3}{ml^2} T_c \tag{7}$$

Here, g denotes the gravitational acceleration, l the length of the pendulum, and m its uniformly distributed mass. The low dimensionality of this system also allows for better visualization of the state space, the actor's policy, and the critic's value function.

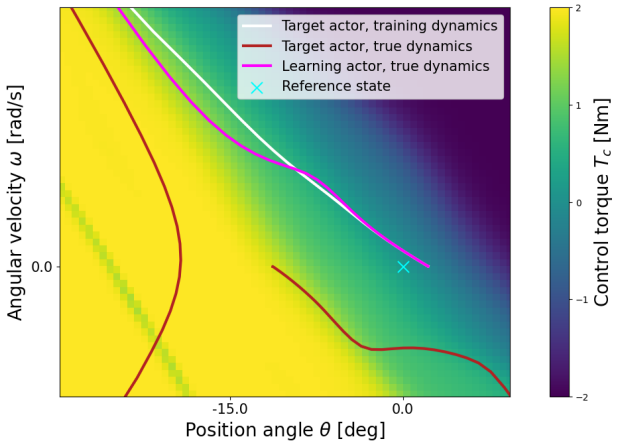
IV. Results & Discussion

The performance of the corrective algorithm is evaluated by comparing trajectories under different operating conditions. The objective is to demonstrate that the learning actor can be online fine-tuned such that its trajectory in the real world converges toward the trajectory of the target actor in the training environment. To this end, the following three trajectories are considered:

1) **Target actor, training dynamics** - This is the trajectory generated by the target actor in the training environment. It is the trajectory which the corrective algorithm strives to approach.
2) **Target actor, true dynamics** - This is the trajectory that would have been generated by the target actor in the real-world environment, had no corrective algorithm been used to account for the mismatch between the real-world dynamics and the dynamics of the training environment.
3) **Learning actor, true dynamics** - This is the actual trajectory generated by the learning actor in the real-world environment. The learning actor attempts to mimic the target-actor trajectory under the training dynamics, enabled by the IHDP-informed update rule shown schematically in Equation 6.



(a) The three trajectories with the actor policy shown as background. Dotted lines indicate successive state transitions and should not be interpreted as the actual trajectories.



(b) Zoomed-in view of Figure 8a around the stationary upright position. The corrective algorithm enables the system to reach the same state as the target actor in the training environment, namely the stationary upright equilibrium.

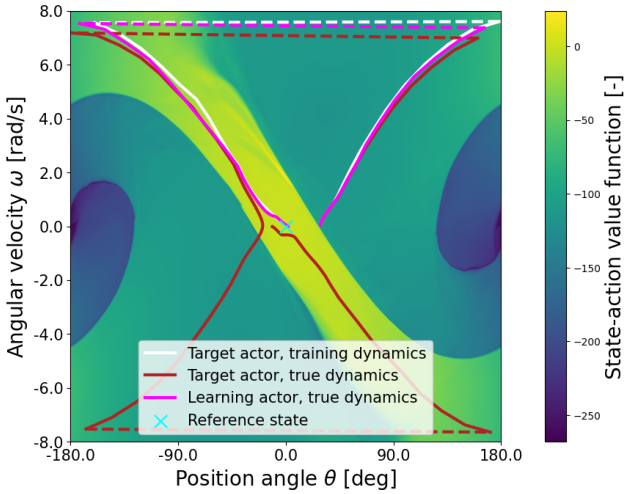
Figure 8. The three trajectories obtained using an offline-trained SAC-CAPS agent. The background shading represents the corresponding actor policy.

A small constant torque, $T_{rw} = -0.6$ [Nm], was added to the action chosen by the learning actor to simulate the real-world aerodynamic phenomena, which were not part of the training environment. The magnitude of this bias torque T_{rw} is 30% of the maximum allowed control torque in the pendulum environment. The different trajectories are shown in Figure 8.

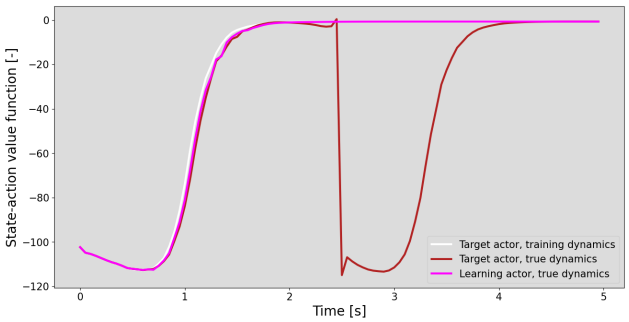
The proposed corrective algorithm closely replicates the target actor's trajectory in the training environment, converging to the same upright stationary equilibrium. In contrast, the target actor under real-world dynamics follows a different trajectory and does not reach the upright position. These trajectories also reveal notable differences in the state-value function histories, as shown in Figure 9.

The inability of the target actor under real-world (true) dynamics to reach a near-equilibrium state during its first pass resulted in a sudden decrease in the state value. Had the corrective algorithm not adapted sufficiently quickly and instead followed a similar trajectory, this detour would not have adversely affected the online learning process. However, in the context of fault-tolerant adaptive control, such a detour could be critical or even catastrophic in a real flight emergency situation.

Finally, Figure 10 shows the condition numbers of the $A^\top A$ matrix associated with the least-squares fit of the incremental model over time. The figure demonstrates that even in a relatively simple environment, such as the Gym pendulum environment, the condition numbers may become excessively large.



(a) The three trajectories from Figure 8 with the critic's state-value value function as the background. Dotted lines indicate successive state transitions and should not be interpreted as the actual trajectories.



(b) The state-value function of the three trajectories over time.

Figure 9. The same three trajectories as in Figure 8, with the critic's state-value function as the background.

V. Conclusion

This paper has discussed the simulation-to-reality gap and proposed a novel corrective algorithm to bridge it. The proposed method enables safe online fine-tuning of a pre-trained deep reinforcement learning agent. This is achieved through a target actor, which the learning actor attempts to track based on its referential state transitions. It was also shown that the CAPS implementation should be used by default, and several approaches for mitigating excessively large condition numbers were discussed.

However, due to the strong ill-posedness of the incremental model's underlying linear system, successful deployment in a drone environment was not

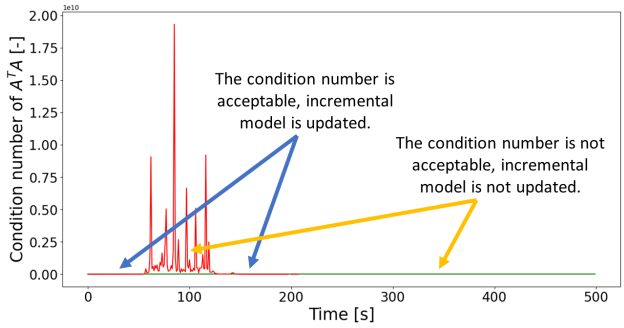


Figure 10. Condition number (red) of the $A^\top A$ matrix over time. The green line represents the threshold for updating the model. The model is not updated if the condition number is too high.

achieved. Consequently, the performance of the method was demonstrated in a pendulum environment as a proof of concept. Future work will focus on more robust differentiable approximations of the plant dynamics, as well as the use of local function approximators instead of neural networks. This is expected to localize online updates around visited states and reduce unintended interference with the pre-trained policy in other regions of the state space.

Acknowledgments

I would like to thank my supervisor, Dr. Erik-Jan van Kampen, first for giving me the opportunity to conduct my research under his supervision, and second for providing me with the freedom to explore this field at my own pace. Without the many consultations we had, this work would not have been possible.

Special thanks also go to Ir. Erik van der Horst and Eduardo Jerez, B.Sc., for their support in the ongoing development of the transitioning vehicle used to test the proposed controller.

The author acknowledges the use of computational resources of the DelftBlue supercomputer, provided by the Delft High Performance Computing Centre (`https://www.tudelft.nl/dhpc`) [18].

Finally, I would like to thank Dr. Dennis Palagin for his guidance regarding the operation and correct use of the DelftBlue supercomputer.

References

[1] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, A Bradford Book, Cambridge, MA, USA, 2018.

[2] Qin, Y., Zhang, W., Shi, J., and Liu, J., "Improve PID controller through reinforcement learning," *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*, 2018. doi: 10.1109/GNCC42960.2018.9019095.

[3] Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M., "Control of a Quadrotor With Reinforcement Learning," *IEEE Robotics and Automation Letters*, 2017. doi: 10.1109/lra.2017.2720851.

[4] Deshpande, A. M., Kumar, R., Minai, A. A., and Kumar, M., "Developmental Reinforcement Learning of Control Policy of a Quadcopter UAV with Thrust Vectoring Rotors," , 2020. URL `https://arxiv.org/abs/2007.07793`.

[5] Li, H., Sun, L., Tan, W., Jia, B., and Liu, X., "Switching Flight Control for Incremental Model-Based Dual Heuristic Dynamic Programming," *Journal of Guidance, Control, and Dynamics*, 2020. doi: 10.2514/1.G004519.

[6] Zhou, Y., Van Kampen, E., and Chu, Q., "Incremental Model Based Heuristic Dynamic Programming for Nonlinear Adaptive Flight Control," *Proceedings of the International Micro Air Vehicles Conference and Competition*, 2016.

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

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

[9] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," , 2017. URL `http://arxiv.org/abs/1707.06347`.

[10] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "OpenAI Gym," , 2016. URL `https://arxiv.org/abs/1606.01540`.

[11] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N., "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, 2021.

[12] Xu, J., Du, T., Foshey, M., Li, B., Zhu, B., Schulz, A., and Matusik, W., "Learning to Fly: Computational Controller Design for Hybrid UAVs with Reinforcement Learning," *ACM Transactions on Graphics*, 2019. doi: 10.1145/3306346.3322940.

[13] Smeur, E., "Incremental Control of Hybrid Micro Air Vehicles," Ph.D. Thesis, Delft University of Technology, 2018.

[14] Morales, M., *Grokking Deep Reinforcement Learning*, Manning Publications Co., Shelter Island, NY, USA, 2020.

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

[16] Mysore, S., Mabsout, B., Mancuso, R., and Saenko, K., "Regularizing Action Policies for Smooth Control with Reinforcement Learning," , 2020. URL `https://arxiv.org/abs/2012.06644`.

[17] Zhou, Y., "Online reinforcement learning control for aerospace systems," Ph.D. Thesis, Delft University of Technology, 2018.

[18] Delft High Performance Computing Centre (DHPC), "DelftBlue Supercomputer (Phase 2)," `https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2`, 2024.