Potential and challenges of reinforcement learning for flow control

Romain Paris

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAX119

Thèse de doctorat

Potential and challenges of reinforcement learning for flow control

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat: Mécanique des fluides et des solides, acoustique

Thèse présentée et soutenue à Meudon, le 2 décembre 2022, par

ROMAIN PARIS

Composition du Jury :

Nicolas Thome Professeur, Sorbonne Université	Président
Laurent Cordier Directeur de recherche CNRS, Institut Pprime	Rapporteur
Lionel Mathelin Chargé de recherches CNRS, LISN	Examinateur
Georgios Rigas Associate Professor, Imperial College London	Examinateur
Julien Dandois Directeur de recherche 2, ONERA (DAAA/MASH)	Directeur de thèse
Samir Beneddine Ingénieur de recherche, ONERA (DAAA/MASH)	Encadrant
Franck Hervy Ingénieur, Direction Générale de l'Armement (AID)	Invité

Contents

Remerciements

Ce doctorat a été en partie financé par une bourse de l'Agence Innovation Défense (DGA).

Parmi ceux qui ont le courage d'utiliser ce cale-porte / dessous-de-plat / objet décoratif de façon détournée, c'est-à-dire en lisant son contenu, je tiens à remercier Laurent Cordier et Nicolas Thome, les deux rapporteurs de mon jury, qui m'ont donné de multiples pistes d'amélioration du présent objet. Merci également à Lionel Mathelin, Georgios Rigas et Franck Hervy pour les riches discussions que nous avons eues durant la soutenance. Celles-ci m'ont offert de nombreux chemins à explorer et des perspectives de recherche toutes plus excitantes les unes que les autres.

"Ah! La recherche, du temps perdu!" Ce cher Marcel Proust m'avait pourtant prévenu... Cette mise en garde ne m'a (malheureusement) pas empêché d'accepter la proposition de thèse de Julien et Samir, qui m'ont présenté cette thèse comme un défi: "partir de rien et voir ce qu'on peut faire". Me voici donc trois ans plus tard, et, n'ayant pas réussi à entraîner l'agent qui devait écrire le présent manuscrit à temps, j'ai du "annoter un peu plus de données"[1]. Ceci me permet donc de vous remercier tous les deux, pour tout ce que j'ai appris avec vous, pour votre énergie et votre passion scientifique communicative. Merci à toi Julien pour ton engagement constant à me guider dans cette thèse, à relire toutes mes productions écrites pas toujours au top avec ce perfectionnisme légendaire[2] que tout le monde te connaît mais toujours dans la bienveillance. Merci à toi Samir, pour toutes les discussions formelles et informelles que nous avons autour de cette thèse qui m'ont permis de maintenir un solide facteur 10 entre le nombre d'idées au début d'une réunion et le nombre de pistes à explorer à la sortie. J'ai peut-être perdu quelques neurones (des vrais hein...) lors de ces réunions mais j'ai appris à réfléchir en prenant du recul, en faisant des liens avec d'autres disciplines et en essayant d'adopter d'autres points de vue. Cela fera certainement de moi quelqu'un qui cherche mieux (à défaut de trouver) et je vous en remercie infiniment.

Quelques mots ensuite pour remercier ceux que j'ai plus vus que ma famille et mes proches et qui, avec le temps sont devenus des amis. Merci à toi Mathieu, avec qui je partage depuis mon arrivée à l'ONERA un bureau (et des canards). Merci de m'avoir aussi bien accueilli à l'ONERA, mis le pied à l'étrier en stage (sur des codes CFD dont le mystère du fonctionnement n'a d'égal que la vacuité de la documentation). Merci d'avoir partagé tous ces repas, ces apéros, ces discussions, sur tout, sur rien et surtout sur rien.

Merci aussi à tous mes compagnons de ~~galère~~ thèse, Pierre N. et Xavier C. avec qui les JDD m'ont donné matière à rire (plus qu'à pleurer). Je compte sur vous pour être les prochains sur la liste des soutenances! Merci à tous les anciens thésards, Johann, Luis, Jean Lou, Quentin, Diogo, Lucas, Tristan et Markus avec qui j'ai partagé de nombreuses pauses-café et fous rires et dont nous avons hérité d'une cafetière. Merci et désolé aux futurs docteurs et membres de la horde: Julian, Arthur P., Arthur V., Mathieu S., Michele, Carmen, Clément, Loïc et Alexandre. Merci pour votre (inconscient) optimisme, vos discussions méridiennes à table ou au café ou pendant les footings. Et désolé. Désolé pour la cafetière qui a fini par rendre l'âme après les milliers de cafés servis à des générations de thésards.

Merci à tous les membres de l'équipe MASH (Missiles, Aéronefs de combat, Stabilité, Hypersonique) pour qui mon appartenance à l'unité est certainement un sujet de recherche ouvert, merci pour votre accueil. Merci aussi à Tanya pour m'avoir bien aidé pour les diverses démarches administratives et pour l'organisation de la soutenance. Merci à la DSI qui a fermé les yeux sur ma

[1] Euphémisme couramment employé en AI pour dire qu'on a tout fait à la main pour présenter une performance acceptable.

[2] Il se dit que Google Scholar source ses citations dans la bibliographie des papiers de Julien

non-allocation de 11 millions d'heures CPU. Merci à toi Jean-Pierre pour ta bonne humeur, tes barbecues estivaux et tes anecdotes Onériennes toutes plus incroyables les unes que les autres.

Merci aux fidèles canards, dont l'énumération nominative serait inutile et fastidieuse, pour leur constante bienveillance dans la résolution de bugs informatiques en tous genres. Pêle-mêle: merci au café, à StackOverflow[3], au chocolat[4] ainsi qu'à arXiv et Scih*b. Au rang des remerciements protocolaires toujours, un grand merci à l'École doctorale IP-Paris qui a su, malgré les circonstances exceptionnelles de ces années covid, garder son exemplaire constance dans la médiocrité de son suivi et de son soutien. Ce niveau demande un désengagement de tous les instants et ne saurait qu'annoncer d'autres records à l'avenir.

Enfin, je ne peux conclure sans penser à ma famille et mes proches sans qui j'aurais certainement abandonné cette thèse, en particulier à mes parents pour leur soutien indéfectible, l'éducation et la curiosité scientifique que j'ai reçue. Merci aussi à ma sœur Florence pour son soutien durant ces trois années. Je te souhaite d'aller loin dans tes projets et de t'y épanouir, tu peux compter sur moi comme j'ai pu compter sur toi pendant cette thèse. Merci à ma grand-mère, à mes cousins, amis de l'X (Martin, Louise, Axel et tant d'autres), amis de classe prépa[5] et de lycée qui, malgré le temps et la distance, ont toujours gardé ce lien et ont su comprendre mes absences à tant d'invitations ces trois dernières années.

[3]Pour avoir répondu des centaines de fois avec une invariante patience aux mêmes questions idiotes.

[4]noir 70%

[5]on ne s'appelle pas "la secte" pour rien!

Acronyms

AI	Artificial Intelligence 156
API	Application Programming Interface 42, 43, 45, 50, 54, 155, 181
CFD	Computational Fluid Dynamics 43, 46, 47, 50, 65, 87, 152, 155
DMD	Dynamic Mode Decomposition 16, 20, 48
FCNN	Fully Connected Neural Network 33, 36, 37
GAE	Generalized Advantage Estimation 28, 30
GP	Genetic Programming 20, 39–41, 47, 179
HPC	High Performance Computing 49, 155
KS	Kuramoto-Sivashinsky 10, 47, 52, 67, 72, 130, 145, 146, 148, 150
LGP	Linear Genetic Programming 11, 14, 39, 52, 67, 109, 155, 179, 181
LTI	Linear Time Invariant plant 15, 180
MDP	Markov Decision Process 21, 27, 151
MIMO	Multi-Input-Multi-Output 10, 17, 18
ML	Machine Learning 9, 20, 37, 42, 43, 67
MPC	Model Predictive Control 20
MPI	Message Passing Interface 49, 99, 178
NN	Neural Network 11, 20, 29, 31–34, 36–38, 46, 68, 127, 140, 149, 177
PDE	Partial Differential Equation 29, 156
PETS-MPC	Probabilistic Ensemble Trajectory Sampling for Model Predictive Control 30, 72, 136
PID	Proportional-Integral-Derivative (controller) 18, 22
POD	Proper Orthogonal Decomposition 9, 16, 17, 38, 48, 113
PPO-CMA	Proximal Policy Optimization with Covariance Matrix Adaptation 26, 29, 68, 72, 88, 144, 150
RL	Reinforcement Learning 9–11, 20, 21, 28, 29, 39–41, 43, 47, 52, 54, 57, 64, 67, 68, 72, 83, 88, 109, 113, 122, 123, 140, 143, 144, 149, 151, 152, 155, 179, 181
SGD	Stochastic Gradient Descent 35
SGL	Stochastic Gating Layer 113, 114, 126
TD	Temporal Difference 28

Chapter 1

Introduction

Most of the ground work concerning the currently emerging applications of Machine Learning (ML) dates back from the 1950s. Yet at that time, the proposed methods and algorithms required computational resources that were by far not available. One had to wait for the turn of a new century to see the rise of super-computers capable of handling such calculations in a moderate amount of time and for a reasonable financial cost. Since then, proofs of concept demonstrating the undeniable potential of machine learning emerged in a wide range of domains. Still, real-world (and useful) applications remain confined to a reduced number of use-cases such as natural language processing or computer vision. Scaling-up toward the levels of complexity of reality in the majority of these domains appears a major challenge.

Ultimately all these machine learning approaches generally come down to regression problems. Thus, the line separating these methods from classical mathematical optimization is at most blurry or may even be better represented as a spectrum of techniques, the major differing traits being the degree of linearity of the problem, and most importantly, the need for generalization (i.e. robustness) on unseen data. It is then better suited to consider machine learning as a (non-linear) extension of decades-old optimization methods such as linear regression, principal component analysis[1] or interpolation methods, rather than a standalone and miraculous set of techniques as the current keen interest for the matter often suggests.

Originally designed to automate industrial monitoring, control went from self-regulated steam engines to increasingly complex applications, even giving birth to modern computers. In fluid mechanics, control aims at altering the flow characteristics in order to improve its behavior regarding a pre-defined objective such as drag reduction, flow stationarization or mixing enhancement. Even nowadays, industrial applications of flow control remain mostly limited to passive control. This stems from the potentially high complexity of a flow both in terms of degrees of freedom than concerning its dynamics, which generally renders other forms of control hard to compute, energy inefficient, or not robust enough. Many recent control techniques are formulated as an optimization problem over a linearized model of the system and rely on some powerful linear algebra approaches. This linear modeling may unfortunately become the Achilles' heel of these methods when applied on non-linear flow dynamics.

While proposing a different paradigm, Reinforcement Learning (RL) is a type of machine learning that fits adequately to control problems. Where traditional linear control methods need to embrace and/or model the system's dynamics, RL works by trial-and-error to derive increasingly optimal control laws (called policies). In doing so, RL algorithms break free from any consideration of

[1] also referred to as Proper Orthogonal Decomposition (POD)

linearity of the dynamics. Coupled with deep learning, capable of embodying non-linear Multi-Input-Multi-Output (MIMO) relations, this approach comes back to the temporal domain, long deserted by traditional methods generally working in the frequency space. In a way, RL trades the power of linear algebra for an increased flexibility.

Better flow control would yield significant edges for overall performances (reduced drag, increased operational speeds), safety (reduced structural loads, increased maneuverability), stealth (reduced acoustic or infrared footprint) and environmental footprint of multiple industrial domains (cleaner and more efficient combustion or reduced consumption). Even if these gains are marginal, they would yield a significant impact thanks to the size of the economic sectors they concern.

This body of work aims at assessing the proficiency of RL concerning flow control, identifying emerging challenges specific to fluid mechanics and proposing innovative methods tackling these issues. This thesis, alongside others, initiates the study of machine learning methods as enablers of better decomposition, analysis, control or generative methods for fluid mechanics. The current work thus aims at bridging the gap between innovative training algorithms that demonstrate impressive performances on sand-box test-beds and more complex flow control test cases for which existing solutions do not behave satisfactorily, data is costly and/or scarce and systems' dynamics are of high-dimension and non-linear. These new methods are to be compared with existing ones, both in terms of efficiency and computational costs. Taking a step back, this study participates in the global effort to scale ML applications up to gradually reach the complexity of the real world.

The present manuscript is organized in the following way:

- A first chapter reviews the state of the art on the topics of flow control, reinforcement learning, deep learning (i.e. the use of neural networks) and genetic programming (a possible alternative to reinforcement learning).
- The following chapter introduces the pre-existing methods and tools used for this work as well as the body of code developed to support the multiple studies led on various test cases and tackling different issues of flow control. These test cases (numerical setup, control setup as well as uncontrolled dynamics) are also presented in this part.
- The main results are presented in the three following chapters:
  - RL-based flow control results on different test cases are presented in chapter 4. Auxiliary studies are led to assess multiple aspects of these control laws and compare them with simpler control approaches.
  - Strategies aiming at reducing the number of flow observations while preserving performance are introduced in chapter 5. These are tested on the cylinder test-case.
  - Chapter 6 introduces a general method proposed in order to reduce the number of actuators. Four different flavors implementing this method are detailed and compared on the Kuramoto-Sivashinsky (KS) test-case as well as on the NACA test-case. Early transposition attempts of this method to model-based RL are also discussed.
- Before presenting the conclusions and perspectives of this work, a few selected challenges are discussed in chapter 7. They are found to be pivotal and interesting issues in most of the studies led in this work.
- At last, appendices developing ancillary concepts or discussions can be found at the end of the present manuscript.

Chapter 2

State of the art

This chapter details the current state of the art on the main domains associated in this thesis: flow control and (deep) reinforcement learning. A non-exhaustive description of control for fluid mechanics is first carried out. Then, Reinforcement Learning (RL) is introduced, followed by a focus on Neural Networks (NNs) that are used in deep RL. At last a short glance is given at Linear Genetic Programming (LGP).

2.1 Control in fluid mechanics

2.1.1 General context and motivations

Flows around an airborne vehicle, a ship or into the piping of an industrial plant often display undesired characteristics, that common practices in design optimization cannot generally get rid of. Using control, one can resort to leveraging the flow to, by small forcings, alter its configuration and hopefully drive it toward more desirable flow states.

Automatic control is a child of the industrial revolution, gaining in importance when the first needs for self-regulated machines arose. The steam engine then the electrification and the telephone gave a strong boost to the development of the theoretical bases of this domain. It has since been used in an ever-increasing number of fields requiring machinery, including fluid mechanics and all its related applications. Considering the overall tendency of flows to display non-linear dynamics and/or to grow instabilities than can end up in chaos, controlling a flow is generally a tricky challenge. Thus, industrial applications directly using flow control are, still as of today, rare.

Beyond the interest of theoretically understanding the dynamics of flows, practical control goals cover drag reduction, lift increase, turbulent transition control (delay or trigger), mixing enhancement, noise or pollutants emissions reduction or flow separation control. All these goals can either be considered as safety and airworthiness issues or performance improvement enablers, hence a significant interest for the research in flow control.

2.1.2 Passive control

Passive control is not, *per se*, considered as an automatic control category, but one commonly refers to this term when talking about small fixed devices altering flow dynamics. These small design alterations hence use the very idea of control: leveraging the flow power to amplify the effect of a

small forcing. Passive control methods make up the majority of flow control cases at the industrial scale, thanks to their simplicity and robustness.

Surface modifications generally aim at modifying the behavior of the boundary layer. They consist either in the use of patches of porous media [42, 34] in order to damp boundary-layer fluctuations by momentum transfer to the absorbing media, or in surface roughness treatments or coatings [70, 13] that force boundary layer transition to turbulence.

Larger pointwise devices are also used to force the flow in strategic locations. Vortex generators [217, 218, 111] are small fins attached to or close to lifting surfaces in order to generate a vortex and/or deflect the flow close to walls, with the effect of delaying flow separation. Vortex-induced vibration is another well studied issue that can affect industrial chimneys or suspension bridge cables. Numerous solutions consist in reducing the coherence length of the load fluctuation [256, 176, 188] by addition of roughness elements to the main structure. Strykowski & Sreenivasan [234], Strykowski & Hannemann [233] and later Marquet *et al.* [152] studied the effect of the addition of a small secondary slender structure carefully located in the unsteady wake and demonstrated positive effects on the stabilization of the flow. The same technique was successfully used by Illy *et al.* [103] to control transonic flow oscillations over a cavity. All these solutions yield an effect that is dependent on the flow state and its interaction with the device, but the latter is steadily applied or fixed on structures and does not require any energy input, hence the qualification of "passive" control.

2.1.3 A quick tour of actuators

Before diving into the different active control methods, let us quickly describe the a wide variety of actuators that can be implemented on experimental setups or simulated. Depending on the flow regime (mainly the Reynolds and Mach numbers), the required power, bandwidth or robustness, technologies will be preferred over other. In their 2011 review on *Actuators for Active Flow Control*, Cattafesta III & Sheplak [45] propose to classify technologies into four categories: fluidic suction/blowing actuators, moving surfaces or objects, plasma-based actuators and body-force actuators such as magnetohydrodynamic actuation.

Among fluidic actuators, one can distinguish zero-net-mass-flux (ZNMF) actuators [79] from valves and oscillators, the latter requiring an external flow bleed contrary to the first. ZNMF actuators, also called "synthetic jets", thus cannot provide actions with a continuous component since it would violate their time-averaged null mass-flux. Fluidic actuators are generally limited by their power and frequency response, their use being generally limited to low-speed flows.

Moving surfaces are a well developed technique (thanks to its prevailing use for aircraft) but may suffer from flow loading and may be limited in frequency response (which is linked to the dimensions and inertia of the moving parts). At the other end of the spectrum in terms of frequency and displacement, the progresses in piezoelectric actuation has enabled miniaturization and thus gains in resonant frequency but at the expense of a decreased power. Moving surfaces are in addition hard to simulate, since, using traditional simulation methods, mesh adaptation or even re-meshing would be necessary at every step of the control. Specific methods such as the immersed boundary method (IBM) or chimera meshing [156] can be used to overcome this issue, but faithful simulations should still take the load induced by the fluid on the control surfaces into consideration, thus turning a fully-fluidic problem into a much more complex fluid-structure interaction.

Plasma-based actuators are mainly dielectric-barrier-discharge (DBD) plasma actuators, where the phenomenon of electrical breakdown is used to locally accelerate the flow by momentum transfer between ionized air molecules and the rest of the flow. This method can be, to a certain extent,

considered as a body-force method. This technology has been investigated by Wang *et al.* [244] who concluded that DBD plasma actuators could "replace movable control surfaces of aircraft" such as Gurney flaps. Yet and contrary to moving parts, plasma actuators generally benefit from fast response but may require high voltage sources and are limited by their gain (energetic output on the flow compared to the primary energy used to power the device). Sparkjets, whose potential for supersonic flows has been assessed by Cybyk *et al.* [59] or localized arc plasma actuators [203] leverage plasma discharges to generate a synthetic jet without moving parts and thus may be relevant for high-speed applications.

Finally, other body-force actuation methods can only be applied on very specific flows where a significant portion of the it is responsive to electro-magnetic fields (such as ionic propulsion for satellites). Thus, excepted for rare proofs of concepts, this method generally discarded for control in conditions of continuous flow (very low Knudsen numbers).

As mentioned, the issue of properly simulating the effects of actuators is crucial for the successful transposition of control laws from numerical simulation to experiments. Yet, in the context of the current study, limited attention is paid to the level of realism of actuator modeling since, contrary to traditional control methods, the level of maturity of deep-learning-based control is rather low. Thus, the main concern is first to provide numerical proofs of concept, demonstrating the mathematical feasibility of such control laws. Experimental demonstration and real-world prototypes are expected to be studied later.

2.1.4 Open-loop control

Open-loop control consists in implementing a pre-computed, time-varying control law or action sequence on the system. Upon forcing, no feedback from the system, quantifying the effect of the action, is taken into account. This assumes that the control objective, toward which the system is driven, is either time-independent or at least does not depend on the initial phase of a periodic system starting state.

The potential of open-loop control has mainly been studied on flow separation. The effects of control on airfoil flow separation using a wide variety of forcings were extensively studied with the goal of delaying stall. Seifert *et al.* [215], Wygnanski [259], Seifert & Pack [216] demonstrated that suction-side oscillatory blowing on the hinge of a flapped airfoil could have benefits on performance (such as increased lift, reduced drag and stall delay). The effects on unsteady blowing/suction control on vortex lift (also called super-lift by the authors) was also investigated by Wu *et al.* [258], with the aim of disrupting the traditional elongated airfoil profile in favor of more complex, multi-core lifting devices. The stall-delaying effect of zero-net-mass-flux suction-side blowing was assessed by Amitay & Glezer [2] on moderate Reynolds (3.1×10^5) stalled airfoil flows who demonstrated a complete flow reattachment when forcing frequencies were an order of magnitude larger than the natural shedding frequency. The delay of dynamic stall (where the airfoil's angle of attack varies periodically) may enable an increased potential for both rotorcraft and highly maneuverable jets. It was studied both in a open-loop and closed-loop framework by Post & Corke [180] using plasma actuators. Investigations on the duty-cycle showed lift improvements ranging from 4 to nearly 13%. More recently Patterson & Friedmann [178] used open-loop blowing control on both sides of rotorcraft blades to reduce vibratory loads on the rotor hub, computing optimized action sequences using a reduced-order model.

Aside from aerodynamics performance improvement, another goal of reducing flow separation concerns structural fatigue. Vortex shedding occurring from some flow separation regimes yields

unsteady loads on the structures. These can lead to premature fatigue and become a severe airworthiness issue. Thus, even on bluff-bodies, that are not designed to generate lift, wake control is also a matter of concern because wake unsteadiness may lead to unexpected trajectories of the body. Focusing on the generic Ahmed body at Reynolds numbers ranging from $2,000$ to $23,000$, Krajnović & Fernandes [129] as well as Parkin *et al.* [177] studied the effects of the actuation frequency for trailing-edge blowing as a mean to increase the near-wake pressure, thereby reducing drag.

The canonical case of the bi-dimensional cylinder flow was also extensively used for open-loop control whether it is via rotary control [119, 90, 184, 183, 133] and the effects of the actuation frequency and amplitude on the flow, using the Lorentz force to control a low-conducting flow [49], via vibrations [247], uniform blowing/suction [231] or using mixed approaches such as auxiliary rotating rods around the cylinder [221].

Open-loop flow control was also been tested for mixing enhancement or conversely the stabilization of Kelvin-Helmholtz-unstable mixing layers. Unsteady cavity flows also count among the academic cases investigated both from the viewpoints of stability and control. Drawing a parallel with a forced Van der Pol oscillator and leveraging the flow global stability, Sipp [225] studied the effects of both frequency and amplitude of a source-term, gaussian-shaped forcing near the leading edge of the cavity. Arnoult *et al.* [6] led an experimental study of the effects of unsteady blowing from this same location on the flow using magneto-mechanical micro-valves and demonstrated an interesting reduction of the noise generated by the interaction between the unsteady shear layer and the downstream edge of the cavity. Mixing layer enhancement, notably using DBD-plasma actuators has been investigated by Singh & Little [222]. On a similar case Li *et al.* [139] used Linear Genetic Programming (LGP) to optimize the open-loop control parameters of the forcing. Extending the scope to planar and round jets, Corke & Kusek [55] assessed the potential acoustic disturbances to drive a jet using its resonances. This idea was later leveraged by Smith *et al.* [230] using zero-net-mass-flow actuators and Suzuki *et al.* [236] with flap actuators to force jet bifurcations.

Most of the studies rely on a parameter search of the amplitude and frequency of a periodic forcing action. This means that, on the long run, the control action may be strong enough to impose its phase and frequency to the whole system, irrespective of its uncontrolled dynamics (such as its phase). Not being able to adapt to the current state of the system may end up in unnecessarily strong and potentially non-robust control actions, being energy inefficient and contravening the genuine idea of automatic control consisting in leveraging the flow power to yield large effects thanks to small forcings.

2.1.5 Linear model reduction

Before the generalization of modern computers, control was limited to low-dimensional input/output systems and most of its methods relied on linear algebra. Linear closed-loop control is thus at the heart of automatic control and most of its base concepts come from this paradigm. The majority of control design methods rely on a two-step process, that first models the system linearly then designs an optimized feedback control law.

2.1.5.1 Modeling the system as a linear plant

This main idea is to consider that the system dynamics can be "reasonably" well approached by a linear model and then take advantage of linear algebra to derive linearly optimal control laws.

Mathematically it reads:

$$\frac{d\underline{q}}{dt} = \mathcal{N}\left(\underline{q}\right) + \underline{f},$$

where $\mathcal{N}$ is the dynamics operator of the system, $\underline{q} \in \mathcal{R}^n$ its full state and $\underline{f}$ the effects of the control forcing on dynamics. In the case where the control goal is to stabilize the system around a steady state $\bar{\underline{q}}$, one can hypothesize that the current state $\underline{q}$ is "close-enough" to the control target such that $\underline{q}' = \underline{q} - \bar{\underline{q}}$ is infinitesimal. Thus, the linearization of the dynamics reads:

$$\frac{d\underline{q}'}{dt} = \underline{\underline{A}}\underline{q}' + \underline{f}, \tag{2.1}$$

where $\underline{\underline{A}} = \left.\frac{\partial \mathcal{N}}{\partial \underline{q}}\right|_{\bar{\underline{q}}}$ is the Jacobian operator of the dynamics around steady state $\bar{\underline{q}}$.

Once linearized, one can use the state-space model representation to fully describe the control problem, getting a Linear Time Invariant plant (LTI) representation:

$$\begin{aligned}\frac{d\underline{q}'}{dt} &= \underline{\underline{A}}\underline{q}' + \underline{\underline{B}}\underline{u},\\ \underline{y} &= \underline{\underline{C}}\underline{q}' + \underline{\underline{D}}\underline{u},\end{aligned}$$

where $\underline{u}$ is the control input and $\underline{\underline{B}}$ is a linear model of the input command on the flow: $\underline{f} = \underline{\underline{B}}\underline{u}$. $\underline{y}$ denotes the output vector or the measurement on the system, while $\underline{\underline{C}}$ and $\underline{\underline{D}}$ model the measurement function, $\underline{\underline{D}}$ the "feedthrough matrix" generally being null.

In cases where the control objective is not steady (e.g. driving the flow toward a non-linearly saturated limit cycle), one can resort to Floquet's theory to model the problem.

2.1.5.2 Observability and controllability

Observability and controllability are two major concepts of the control theory [118]. Controllability is the ability to place the eigen values of a linearly fed-back ($\underline{u} = -\underline{\underline{K}}\underline{y}$ in the *s-domain*) system anywhere in the complex space. Observability measures how the internal state $\underline{q}$ of the system can be deduced from observations $\underline{y}$.

Applied on linear(ized) system dynamics, observability and controllability are related to the rank the observability $\mathcal{O}$ and controllability $\mathcal{C}$ matrices:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{v-1} \end{bmatrix} \quad \mathcal{C} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix}.$$

The system is then observable if $\text{rank}\left(\mathcal{O}\right) = n$ (full column rank). On large systems and using a reduced set of sensors the unobservable space $\ker\left(\mathcal{O}\right)$ is unavoidably large. But one can derive the weaker notion of detectability, that only concerns unstable state sub-spaces. Hence the system will be said "detectable" if all the unobservable sub-spaces are stable and thus do not represent a

"threat" to control. Conversely, the system is said controllable if rank $(\mathcal{C}) = n$ (full row rank). The Cayley-Hamilton theorem[1] then establishes the equivalence between controllability and reachability (i.e. the possibility to reach any state of the system using control), the latter being the relevant indicator on the feasibility of feedback control.

Stabilizability is the dual notion of detectability, the system is "stabilizable" if all the uncontrollable state sub-spaces have stable dynamics. In the case of reduced-order models, preserving both the "detectability" and the "stabilizability" are crucial to a successful closed-loop control of the dynamics.

Introduced by Moore [162], Gramians (W_{obs} and W_{con} resepectively) bring a more precise viewpoint of both observability and controllability:

$$W_{obs}(t) = \int_0^t e^{\underline{\underline{A}}^T \tau} \underline{\underline{C}}^T \underline{\underline{C}} e^{\underline{\underline{A}} \tau} d\tau$$
$$W_{con}(t) = \int_0^t e^{\underline{\underline{A}} \tau} \underline{\underline{B}} \underline{\underline{B}}^T e^{\underline{\underline{A}}^T \tau} d\tau,$$

using the previously introduced state-space matrix notations. One can demonstrate that if W_{obs} is non-singular ($\ker(W_{obs}(t)) = \emptyset$) for any time t, then the system is observable, and reciprocally. The same can be showed for W_{con} and the controllability. The main advantage of Gramians over observability and controllability matrices comes from their ability to provide a mode-by-mode analysis of the controllability and observability. Using an eigen-value decomposition of the Gramians, the largest eigen-values correspond to the most controllable (and reciprocally observable) modes of the system. Moore [162] also showed that one could compute data-driven approximations of these Gramians using the observability and controlability matrices $\mathcal{O}$ and $\mathcal{C}$. This solved the issue of computing the Gramians using their introduced definition which requires the knowledge of $\underline{\underline{A}}$, $\underline{\underline{B}}$ and $\underline{\underline{C}}$ and a possibly large computational overhead.

2.1.5.3 Data-driven model reduction methods and system identification

In the case of flow control, this linear model is a reduced-order model since flows have a number of degrees of freedom ranging from 10^5 and beyond. The reduction step is then not only about linearization but also aims at accurately capturing the dynamics of the flow with a reduced number of degrees of freedom.

Depending whether observations are dense flow fields or sparse pointwise measurements, the reduction can take place on the internal state dimensionality only or also on the number of outputs. Full flow fields may not be tractable for feedback design, thus their main features must be "compressed" into a smaller vector. To this end, multiple studies such as Gerhard *et al.* [76], Bergmann *et al.* [21], Samimy *et al.* [204], Bergmann & Cordier [20] leverage the energy-optimal Proper Orthogonal Decomposition (POD) method to reduce the observed vector and project the dynamics from the full state to a reduced one via a Galerkin projection of the Navier-Stokes equations. Other linear decomposition methods such as the Dynamic Mode Decomposition (DMD) can be used. The latter has the advantage of directly providing the linearized dynamics of the system, but does not guarantee the orthogonality between modes. If the full state observations of the system are tractable, the state-space representation (approximation of the Koopman [127] operator) can even be directly derived using DMD with control (DMDc) as proposed by Proctor *et al.* [182].

[1] i.e. square matrices satisfy their own characteristic equation

If observations are a reduced number of scalar measurements, the need for reducing the number of outputs of the system vanishes but the internal Navier-Stokes-driven state dynamics still need to be linearly reduced. As described by Brunton & Kutz [35], linear model reduction methods have shown tremendous improvements in the last decades. Moore [162] first proposed balanced truncation, which consists in finding a coordinate transform where both controllability and observability Gramians are equal and diagonal and then keeping only the first modes (that are thereby the most controllable and observable). This approach, that captures the most relevant dynamics is yet limited by the need to compute eigen-value decompositions on the linearized dynamics and the Gramians. These computations can become intractable for large systems. Willcox & Peraire [250] later showed that one could compute reduced rank approximation of the Gramians, using Balanced POD (BPOD), to alleviate this issue. BPOD was further improved by Rowley [197] who found a way to by-pass the heavy computation of the Gramians, using a singular value decomposition of the Hankel matrix of the system. Dergham *et al.* [63] used this decomposition to describe the dynamics of both a rounded backward-facing step and an open-cavity flow. BPOD was also used by Tol *et al.* [239] on a turbulent channel flow.

The Eigensystem Realization Algorithm (ERA) was introduced by Juang & Pappa [112] and proposes a balanced representation of the system's dynamics from an impulse response of the system, again using a truncated singular value decomposition of the Hankel matrix of this impulse response. ERA was for instance used by Rizi *et al.* [191] to identify the dynamics of an open-cavity flow. Yet for some systems, it may be practically impossible to simulate or apply a perfect-enough impulse forcing on the system. Observer-Kalman filter identification [113] (OKID) was then proposed to estimate this impulse response from noisy input-output measurements, thus solving this issue.

Auto-regressive models is a class of system identification methods that proposes to model a given measurement as a (here linear) function of previous measurements, control inputs and noise, using the Z-transform. These methods are then implemented in conjunction with standard dimensionality reduction methods for large MIMO systems. Gao *et al.* [75] for instance compared Auto Regressive model with eXternal inputs (ARX) on lift and pitch coefficients measurements of a transonic buffet airfoil flow with BPOD and full-state CFD simulations. On a similar case, Huang & Kim [101] performed ARX on full-state measurements then used balanced truncation. Hervé *et al.* [95] used ARMAX, a modified version of the method handling noise modeling and compared its performances with a standard Galerkin projection of the full-state dynamics on a BPOD decomposition.

2.1.5.4 Resolvent-based approaches

Linear model reduction or system identification aims at describing the linearized dynamics of a system around a fixed (equilibrium) point. Resolvent-based approaches are model-based methods that exploit a linear-response assumption of the system to approximate its dynamics, by directly computing the spectral response of the system to forcing. Using equation 2.1 and performing a Fourier transform in time reads:

$$\begin{aligned} j\omega\underline{\hat{q}}' &= \underline{\underline{A}}\underline{\hat{q}}' + \underline{\hat{f}} \\ \text{or} \quad \underline{\hat{q}}' &= \underbrace{\left(j\omega\underline{\underline{I}} - \underline{\underline{A}}\right)^{-1}}_{\underline{\underline{R}}}\underline{\hat{f}}, \end{aligned} \tag{2.2}$$

where ω is the frequency, $\underline{\underline{I}}$ is the identity matrix and $\underline{\underline{R}}$ is called the resolvent operator. The forcing $\underline{f}$ here comprises the effects of the control input $\underline{\underline{B}}\underline{u}$ but also the possible non-linear part of the internal dynamics of the system as detailed by Beneddine *et al.* [18]. As shown by equation

2.2, the resolvent operator depends on ω and may singularize for some frequencies if the system is globally unstable. Thus, this tool is primarily used for modeling convective instabilities. In addition, the resolvent may provide relevant information about sensor and actuator placement by a study of the optimal forcing maximizing the gain of the system. This approach was used by Jin *et al.* [109, 107], Luhar *et al.* [146] or Yeh & Taira [262].

2.1.6 Linear closed-loop control design methods

Once the system's dynamics are approached and possibly reduced, an optimized feedback control law can be computed. In this framework, the majority of the syntheses is led in the Laplace-transformed complex frequency domain. More details about the Laplace and the formalism used here can be found in section 9.2. Figure 2.1 depicts the standard feedback loop where $G(s)$ denotes the Multi-Input-Multi-Output (MIMO) transfer matrix of the system in the Laplace *s-domain* and $K(s)$ is the control transfer matrix to synthesize.

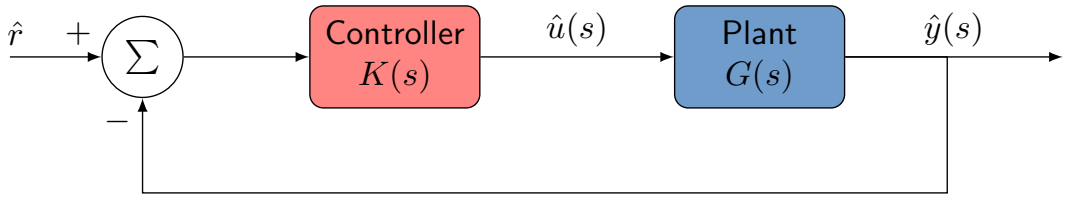


Figure 2.1: Linear feed-back loop schematics.

From this schematics one can derive two important transfers:

$$S(s) = \frac{1}{1 + G(s)K(s)} \quad \text{Sensitivity}$$

$$H(s) = \frac{G(s)K(s)}{1 + G(s)K(s)} \quad \text{Complementary sensitivity (or Closed-loop transfer function).}$$

The sensitivity quantifies the response of the system to an external disturbance $\hat{r}$ (e.g. noise), whereas the complementary sensitivity represents the behavior of the unperturbed measurements $\hat{y}$ under the control feedback K. One of the simplest synthesis methods is loop-shaping. It consists in "manually" tuning the open-loop transfers using a predefined type of feedback K (for instance a PID) in order to place the poles of these transfers. Loop-shaping was for instance used by Brackston *et al.* [30] to tune a law controlling the wake of an Ahmed body.

The first widely used design method, the Linear-Quadratic Regulator (LQR), relies on the optimization of the control with respect to a quadratic cost function and a Kalman filter estimation of the dynamics. The quadratic cost function balances both penalizations between the energy of the internal state of the system and the actuation energy expenditure, using three transfer matrices. The optimal feedback is then provided by the resolution of a Riccati differential equation. This method is limited first, by the difficulty to properly tune the three mass matrices of the cost function and second, by the lack of guaranteed robustness, as demonstrated by the notoriously concise article written by Doyle [67]. In cases where the system dynamics (approached by a Kalman filter) on which the control law is computed are an approximation, the impossibility to guarantee stability on the full-state real-world system may be prohibitive. Loop-Transfer-Recovery proposed to circumvent this issue by "artificially increasing actuator noise to render the estimation process faster" as explained

by Sipp & Schmid [228], but this approach is not suitable for systems such as noise amplifiers. Chen & Rowley [47] used this synthesis method to investigate actuator and sensor placement issues on the Ginzburg-Landau equation. Barbagallo *et al.* [10, 9], Illingworth *et al.* [102] implemented LQR on an numerically-simulated open-cavity flow to successfully stabilize the system, confirming the experimental conclusions of Samimy *et al.* [204] on a similar flow regime. This design method was applied by Lee *et al.* [136] and Yao *et al.* [261] to control a turbulent channel flow. Bergmann *et al.* [21] and Bergmann *et al.* [22] employ the same paradigm to control the wake of a cylinder flow. Gao *et al.* [75] compared LQR synthesis with a simpler pole placement method on a transonic buffet flow around an airfoil and concluded that both sub-optimal control laws were rather similar.

Modern control methods aim at tackling both the issues of optimality and robustness. The main idea underlying the following methods is to formulate an optimization problem on one or multiple transfers (or product/sums of transfers), to solve it in the *s-domain* and to leverage Parseval's theorem[2] to ensure the optimality on the input temporal domain. When using the natural norm of Hardy space H^2 (refered to as H_2-synthesis [85]), the optimization is about minimizing the standard deviation deviation of sensor measurements, that can directly be related to the minimization of the energy of the system in the current linear framework. The standard LQR synthesis is indeed itself a kind of H_2-synthesis. Rizi *et al.* [191] performed an H_2-synthesis of a delay-aware control law on an ERA-obtained reduced-order model of a cavity flow. Manohar *et al.* [150] and Jin *et al.* [108] also employed H_2-synthesis on a BPOD model of the Ginzburg-Landau equation and on a resolvent-based decomposition of a cylinder flow respectively. Tol *et al.* [239] synthesized an H_2-optimal controller to control a turbulent channel flow.

H_∞-synthesis [265] uses the norm of Hardy space H^∞ and is qualified as "robust" synthesis since the optimization is performed with respect to a worst-case scenario. This way robustness is preferred over optimality, closed-loop stability margins being directly related to the H^∞-norm. Thus, in cases where the plant model $G(s)$ suffers from uncertainties, this method provides a conservative method to guarantee the stability of the control on the real-world plant. One can combine these two syntheses using mixed H_2/H_∞ formulations. Bewley & Liu [24] and Bagheri *et al.* [8] compared the performances and robustness of both H_2 and H_∞ syntheses on an unsteady channel flow. This test-case was also used by Jones *et al.* [110]. Henning *et al.* [94] demonstrated the robustness of H_∞ control on the wake an Ahmed body.

These modern approaches optimize the parameters transfer matrix K (the control feedback to design), each term k_{ij} of the matrix K being a tunable transfer function from measurement dimension j to command dimension i. The order of these transfer functions is the order of plant $G(s)$. This can become a matter of concern for large order plants, where the dimension of the optimization explodes. More recently, so-called "structured syntheses" [4] thus proposed to tackle this issue by imposing a pre-defined structure (order, number of poles and zeros) to the transfers of K. This way, the order of the control feedback is controlled and the optimization remains tractable. Leclercq *et al.* [135], Nibourel *et al.* [168] and Jussiau *et al.* [114] experimented mixed H_2/H_∞ structured synthesis respectively on a cavity flow, a supersonic flat plate flow and a cylinder flow respectively.

2.1.7 Non-linear control and model-reduction methods

The main limitation of the previously introduced methods is their reliance on an assumption of (near) linearity of the system to control. Thus, when the system does not behave "linearly enough",

[2]Refer to appendix 9.2 for more details.

these approaches are bound to fail. That is where non-linear control comes in, to handle these systems. In addition, even if the system is close to linear, non-linear control approaches aim at being more energy-efficient than linear ones. The flip-side of the abandon of the linearity assumption of the whole closed-loop system is the loss of most of the linear algebra techniques previously used to synthesize control laws and the robustness guarantees coming with these methods. In particular, non-convexity may cause continuous optimization approaches to be stuck in sub-optimal local minima.

Non-linear versions of auto-regressive models, NARX and NARMAX enable to incorporate a controlled "amount" of non-linearity in the dynamics. These have been for instance used by Dandois *et al.* [60] and Kim *et al.* [122], both to control flow separation on a backward-facing ramp.

The idea of embedding the measurements or full-state snapshots into a larger, richer space to more precisely approximate dynamics has been used by Williams *et al.* [254] who proposed Extended DMD by computing extra observable functions of the state alongside snapshots before performing the singular value decomposition. From this idea also originates the Sparse Identification of Non-linear Dynamical systems (SINDY) proposed by Brunton *et al.* [36], where a dictionary of observables of measurements is used as input of a L_1-penalized optimization fitting the temporal derivative of these measurements by linear combination, aiming at identifying the system's dynamics. Both methods were later extended to accommodate exogenous forcing [253, 37]. SINDY was for instance used with MPC by Kaiser *et al.* [116] on multiple test-cases. The main identified issues of these "lift-up" approaches first relate to the choice of the appropriate dictionary of observable functions which requires *a priori* knowledge on the system dynamics and second to the "curse of dimensionality". Increased input/output sizes may lead to a large number of candidate functions in the dictionaries, thus requiring to tune an exploding number of parameters, that in turn needs more data.

Model Predictive Control (MPC) is a control method where the control law is implicitly defined as the solution to an optimization problem. The derived solution is a sequence of control actions that maximize a performance metric on a closed-loop model of the system. The first action of the sequence is to be implemented in the real-world system while the following ones are generally discarded. Nair *et al.* [165] compared this approach on three test-cases with optimal phase control while Korda & Mezić [128] used MPC on a Koopman linear predictor model. Bieker *et al.* [26] developed NN-based MPC models to perform online control of a fluidic pinball flow. This method is not *per se* non-linear but its paradigm is general enough to be used with a non-linear model. The method is close to reinforcement learning in the sense that one seeks to optimize a control sequence taking long-term effects of control forcing into account.

All the previously introduced methods can be considered as Machine Learning (ML), since they aim at identifying patterns and building models from data. The distinction between traditional control methods and the main focus of the current study is more to be sought toward both the structure of the control law and the training method. The previously introduced methods generally rely on a strongly structured control laws, whose parameters (transfer function poles and zeros, matrix parameters, or linear combination weights) are to be optimized and on training methods that make strong assumptions on the dynamics. The present study aims at evaluating the potential:

1. of less-structured or unstructured approaches whose formulation rely on Neural Networks (NNs) and deep learning in general,

2. of innovative training methods on unstructured (with Reinforcement Learning (RL)) and structured (with Genetic Programming (GP) and RL) control laws,

3. and to identify (and if possible tackle) the issues specific to flow control that arise with these new methods.

2.2 Reinforcement Learning

The reinforcement learning (RL) paradigm models the control problem as a time-discrete interaction loop between an agent (implementing the control law) and an environment (the flow to be controlled). As shown by figure 2.2, the agent takes control actions depending on partial observations of the environment, in a closed-loop fashion. Through iterations of this interaction loop, the agent seeks to optimize its decision behavior in order to maximize a reward, which is provided at each control step by the environment alongside observations.

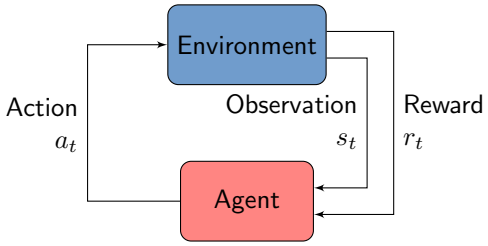


Figure 2.2: The reinforcement learning loop

2.2.1 An optimization problem

2.2.1.1 A Markov Decision Process

This optimization problem can be formalized as a Markov Decision Process (MDP) $\{S, A, T, r\}$ where:

- S is the **state space**, the set of all observable states of the environment;
- A is the **action space**, the set of all authorized actions on the environment[3];
- $T : S \times A \times S \to [0; 1]$ is the **transition function**, defining the dynamics of the environment under control action, quantifying the transition probability $T(s'|s, a)$ from state s to state s' under action a;
- $r : S \times A \times S \times \mathbb{R} \to [0; 1]$ is the **reward function**, defining the probability of obtaining a reward r for the transition $(s, a) \to s'$.

The transition function T being only conditioned by the current state s and action a, the presently defined MDP satisfies the **Markov property**[4]. In the following "$s \sim T$" is a shorthand notation for "$s' \sim T(\cdot|s, a)$" and the "prime" symbol indicates succession (i.e. $\forall\, t\ s'_t = s_{t+1}$).

[3]This set can be conditioned by the state, but this has no impact in the current reasoning.
[4]i.e. it is memoryless: $T(s_{t+1}|(s_t, a_t), \ldots, (s_0, a_0)) = T(s_{t+1}|s_t, a_t)$

2.2.1.2 Defining the control objective

In the context of RL, actions are taken by an agent following a control law. This control law is called **policy** and is generally stochastic: $\pi : S \times A \to [0; 1]$. The policy thus also satisfies the Markov property which theoretically forces it be analogous to a "stochastic non-linear simple gain" function, excluding for instance PID controllers from the possible policies[5]. In the following "$a \sim \pi$" is a shorthand notation for "$a \sim \pi(\cdot|s)$". Starting at $t = 0$ from state s_0, following an infinite controlled trajectory $\tau = ((s_0, a_0), (s_1, a_1), \dots)$, where $\forall t \; a_t \sim \pi(\cdot|s_t)$, one can collect all the rewards $(r_0, r_1, \dots)$ associated with each transition and define the **return** as:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t, \tag{2.3}$$

where $\gamma \in [0, 1[$ is a discount factor, favoring immediate rewards over more distant ones. γ values close to 0 tend to attribute more value to greedy action sequences and conversely, when $\gamma \to 1$ the similar weight is attributed to immediate and distant rewards, favoring "long-term oriented" action sequences. The reward function is by hypothesis, stochastic, and its inputs (state and action) are also the result of a stochastic sampling (on T and π). Thus, the **value** function, namely the expected return of a trajectory τ starting from a given state s and conditioned by policy π is noted as:

$$V^{\pi}(s) = \mathbb{E}_{\tau \sim T, \pi} \left[R(\tau) \right] = \mathbb{E}_{\tau \sim T, \pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right], \tag{2.4}$$

where "$\tau \sim T, \pi$" is a shorthand notation for "$s \sim T, a \sim \pi \; \forall \, (s, a) \in \tau$".

For any given distribution $\mu : S \to [0; 1]$ of starting states s ($s \sim \mu(\cdot)$), the goal of a RL algorithm is to find the optimized policy π^* that maximizes the expected value function over the distribution μ:

$$\pi^*(\mu) = \arg\max_{\pi} \mathbb{E}_{s \sim \mu} \left[V^{\pi}(s) \right] = \arg\max_{\pi} \mathbb{E}_{\substack{s \sim \mu \\ \tau \sim T, \pi}} \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right], \tag{2.5}$$

The corresponding optimal value function being V^* so that:

$$V^*(\mu) = \max_{\pi} \mathbb{E}_{s \sim \mu} \left[V^{\pi}(s) \right]. \tag{2.6}$$

This function V^* exists and equation 2.6 has a unique solution.

2.2.2 The Bellman equation

Bellman [16] demonstrated that:

$$\exists ! \, V^* \text{ so that } \; \forall \, \mu : S \to [0; 1], \; V^* = \max_{\pi} \mathbb{E}_{s \sim \mu} \left[V^{\pi}(s) \right], \tag{2.7}$$

which is a much more general and powerful assertion since, it proves the existence of a single V^* irrespective of the underlying starting state distribution. This is showed by the transformation of

[5] It is nonetheless possible to provide policies with finite difference estimates of integral and derivative terms without breaking the Markov property by artificially embedding previous states into the current observation vector.

equation 2.4 into a self-consistency criterion on the value function V^π. Equation 2.3 can be rewritten considering τ_i, the i^{th}-step shifted trajectory $((s_i, a_i), (s_{i+1}, a_{i+1}), \dots)$:

$$\forall\, i \quad R(\tau_i) = r(s_i, a_i, s_{i+1}) + \gamma R(\tau_{i+1}),$$

as long as $|R_\tau| < \infty$. Thus equation 2.4 becomes:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{\substack{a\sim\pi \\ s'\sim T \\ \tau_1\sim T,\pi}} \left[r(s,a,s') + \gamma R(\tau_1)\right] \\ &= \mathbb{E}_{\substack{a\sim\pi \\ s'\sim T}} \left[r(s,a,s') + \gamma \mathbb{E}_{\tau_1\sim T,\pi}\left[R(\tau_1)\right]\right] \\ V^\pi(s) &= \mathbb{E}_{\substack{a\sim\pi \\ s'\sim T}} \left[r(s,a,s') + \gamma V^\pi(s')\right], \end{aligned} \tag{2.8}$$

Equation 2.8 is the Bellman equation for the value function.

Theorem 2.2.1 (Bellman principle of optimality). $\exists\, s \in S,\ V^{\pi^*}(s) = \max_\pi V^\pi(s) \Rightarrow\ \forall\, (s,a,s') \in S \times A \times S,\ V^{\pi^*}(s') = \max_\pi V^\pi(s')$

Proof. By contradiction, if $\exists\, (s,a,s') \in S \times A \times S,\ V^{\pi^*}(s') < \max_\pi V^\pi(s')$, then the policy $\pi' = \delta_{s,s'} \arg\max_\pi V^\pi(s') + (1-\delta_{s,s'})\pi^*$ has a value function $V^{\pi'}$ so that $V^{\pi'}(s) > V^{\pi^*}(s)$ which contradicts the optimality of π^*. □

In other words, a given policy is optimal if it maximizes $V^\pi(s')$ whatever previous state s and action a are. This way, the decision problem boils down to the optimization of a single state-value with the insurance that all the other likely visited states see their value optimized.

The optimal policy π^* can thus be determined from the globally optimal value function V^* as previously introduced as the argument of the maxima of V^*. This way, π^* is deterministic[6]. In the following, r is supposed to be a deterministic function of the current state and action only, meaning that one writes $r_t = r(s_t, a_t)$. All the formulas and properties introduced earlier still hold.

2.2.3 Model-free RL

Kaelbling *et al.* [115] divide the taxonomy of RL algorithms into two mutually exclusive families: model-based and model-free methods. The first aims at learning the dynamics of the environment and to derive an optimal controller from this approached model. Learning algorithms that do not make use any explicit model of the environment's dynamics are said to be *model-free*. These methods circumvent the difficulties of learning a model, often with the benefit of an increased sample efficiency. One can subdivide the family of model-free RL methods into two approaches of the problem.

2.2.3.1 Value-iteration

Also called Q-learning, it makes use of the **action-value function** $Q^\pi(s,a)$. This auxiliary function is defined as the expected return of a trajectory starting from state s, sampling control actions following π , **excepted** for the first action $a_0 = a$[7]:

$$Q^\pi(s,a) = \mathbb{E}_{s'\sim T}\left[r(s,a,s') + \gamma V^\pi(s')\right] = \mathbb{E}_{\tau_1\sim T,\pi}\left[R(\tau)|s_0 = s, a_0 = a\right].$$

[6] It can be considered stochastic in the rare cases of states where two different optimal actions lead to the same expected return.

[7] i.e. a_0 is deterministically chosen and $\forall\, t > 1\ a_t \sim \pi(\cdot|s_t)$

Thus, by definition of the value function:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}\left[Q^{\pi}(s,a)\right]. \tag{2.9}$$

The action-value function can be expressed as:

$$\begin{aligned}
Q^{\pi}(s,a) &= \mathbb{E}_{\tau\sim T,\pi}\left[r(s_0,a_0,s_1) + \gamma R(\tau_1)|s_0 = s, a_0 = a\right] \\
&= \mathbb{E}_{\substack{s_1\sim T \\ a_1\sim\pi \\ \tau_2\sim T,\pi}}\left[r(s,a,s_1) + \gamma R(\tau_1)|s_0 = s, a_0 = a\right] \\
&= \mathbb{E}_{s'\sim T}\left[r(s,a,s') + \gamma \mathbb{E}_{a'\sim\pi}\left[\mathbb{E}_{\tau_2\sim T,\pi}\left[R(\tau_1)|s_1 = s', a_1 = a'\right]\right]|s_0 = s, a_0 = a\right] \\
\text{thus} \quad Q^{\pi}(s,a) &= \mathbb{E}_{s'\sim T}\left[r(s,a,s') + \gamma V^{\pi}(s')\right] \\
\text{using 2.9 and since} \quad Q^{\pi}(s',a') &= \mathbb{E}_{\tau_2\sim T,\pi}\left[R(\tau_1)|s_1 = s', a_1 = a'\right].
\end{aligned}$$

The objective of Value-iteration is to learn the optimal action-value function $Q^*(s,a)$, using the same self-consistency property ensured by the Bellman equation. The controller π is optimized indirectly by learning its underlying performance metric V. The original value-iteration method is described by algorithm 1.

Algorithm 1 General Value Iteration Algorithm

```
k ← 0
∀ s, a ∈ S × A   Initialize Q_0(s, a) arbitrarily
∀ s ∈ S   V_0(s) ← max_a Q_0(s, a)
while V_k is not converged do
    for all s ∈ S do
        for all a ∈ A do
            Q_k(s, a) ← E_{s'∼T}[r(s, a, s') + γ V_k(s')]
        end for
        V_{k+1}(s) ← max_a Q_k(s, a)
    end for
    k ← k + 1
end while
for all s ∈ S do
    π*(s) ← arg max_a Q_k(s, a)
end for
```

This family of training algorithms is mostly **off-policy**. This means that the training data used at a given step of the training process is collected irrespective of the current optimal policy. In the original Value-iteration algorithm, the update of the action-value function $Q_k(s,a)$ appears inefficient since it is computed for any sate $s \in S$ and action $a \in A$, which becomes intractable from large discrete or continuous action spaces. This is particularly true if a given state s cannot be "replayed" easily, as it is the case for flow environments. Thereby, all Value-iteration algorithms only update $Q_k(s,a)$ on visited state-action pairs and adopt extra/interpolation strategies to estimate non-visited one. Sutton & Barto [235] and other authors consequently demonstrated that these methods usually suffer from a lack of stability, function approximation error and off-policy training data being major causes of failure. Most of the derived algorithms propose to both mitigate these

issues via different techniques, using delayed or differently trained estimators such as Double Q-learning [240] for instance and resorting to smooth approximators to estimate the intractable optimal action $a = \arg\max_a Q_k(s, a)$ in the cases where the action space A is continuous (such as with Deep Q-Network with Normalized Advantage Function estimator (DQN-NAF) proposed by Gu *et al.* [83]). They also manage to take advantage of the off-policy aspects of the method by re-using previously collected data to greatly increase sample efficiency, notably via experience replay [157, 3].

Gueniat *et al.* [84] projected the states of a Lorenz system onto a discrete set of reduced states, similarly to k-means clustering but using a hashing function. They then leveraged tabular Q-learning on the reduced dynamics, encoded as a probabilistic transition matrix between states, to learn a control that forces the state to stay on a predefined "wing" of the attractor. A Deep Q-Network (DQN) was used by Waldock *et al.* [243] to successfully control the trajectory of a variable-sweep wing UAV. Shimomura *et al.* [219] also implemented DQN to experimentally control the flow separation over an airfoil using plasma actuators located near the leading edge.

2.2.3.2 Policy-iteration

On the other hand, Policy-iteration methods explicitly represent the policy π and optimize it using gradient ascent on approximators of the value function V^π. Contrary to Value-iteration, the training is almost always performed **on-policy**, meaning that, at any training step the training data is obtained using actions from the current policy. These methods are thus considered less sample-efficient, since previous data samples (which are off-policy after update) are discarded and never re-used. Policy-iteration algorithms are more stable than value-iteration methods since there are principled, i.e. they directly optimize the policy π which is the goal of the training process. The original Policy-iteration method is described by algorithm 2.

Algorithm 2 General Policy Iteration Algorithm

$k \leftarrow 0$
$\forall s \in S$ Initialize π_0 arbitrarily
$\forall s \in S$ Initialize $V^\pi(s) \leftarrow 0$
while π_k is not converged **do**
  while V^π is not converged **do** ▷ Evaluation step
    for all $s \in S$ **do**
      $V^\pi(s) \leftarrow \mathbb{E}_{\substack{a\sim\pi_i \\ s'\sim T}} [r(s, a, s') + \gamma V^\pi(s')]$
    end for
  end while
  for all $s \in S$ **do** ▷ Improvement step
    $\pi(s) \leftarrow \arg\max_a \mathbb{E}_{s'\sim T} [r(s, a, s') + \gamma V^\pi(s')]$
  end for
end while

The optimization loop of the algorithm decomposes into the evaluation step, in which the policy π is run on the environment to update the value function V^π and the improvement step where the policy π is modified to take actions maximizing the freshly updated value function. Similarly to Value-iteration, as soon as the state space is continuous, of high dimension and/or hard to replay, all algorithms approximate the exploration of the state space, only relying in the data collected

along a control trajectory. To enrich this collected data and as a way to solve the exploration-exploitation dilemma, most of the state-of-the-art methods resort to exploration noise added to the control action during roll-outs to try to progressively explore more interesting regions of the state-space. During the improvement step, these methods cannot explicitly compute the argument of the maximum and thus proceed by smooth gradient ascent. These updates are carefully tuned, using different formulations of the policy gradient, to account both for the possible error of V^π and to the fact that exploration noise is injected during the collection of training data.

TRPO [210] is one of first efficient proposed methods, and was quickly followed by the widely-used PPO [212]. All these algorithms resort to two separate structures to embody the value function and the policy, in an **actor-critic** paradigm. The actor (the policy π) is tasked to derive the best policy and the critic (the value estimator V^π) evaluates the performances of π. In the current study, PPO-CMA [89], a variant of PPO is prominently used. It is more thoroughly described in section 2.2.5.

PPO has been used in numerous studies controlling a confined cylinder flow at various regimes and different actuation layouts [186, 237, 190, 140, 151]. Wang *et al.* [246] implemented this training method on a confined NACA 0012 airfoil flow to improve its lift and Beintema *et al.* [14] managed to reduce the heat transfer on a bi-dimensional Rayleigh-Bénard convection cell, also using PPO. An actor-critic-structured method relying on a structured policy was proposed by Vona & Lauga [242] to stabilize a viscous flow using rotating cylinders.

2.2.3.3 Practical implementation: expected grad-log-prob and baselines

As for value-iteration methods, policy-iteration methods aim at maximizing the expected return $J(\pi_\theta) = \mathbb{E}_{\tau\sim T,\pi}\left[R(\tau)\right]$ but with respect to an explicitly defined policy whose parameters θ are tuned using $\nabla_\theta J(\pi_\theta)$. In the following, J is the generic notation used for the optimization objective instead of V.

Two practical issues arise here: getting an expression of the policy gradient that can be approached using a stochastic estimator computed on collected trajectories and ensuring that this estimator is unbiased and accurate enough to guarantee policy improvement. The first problem is addressed by the derivation of the "Expected grad-log-prob" expression of $\nabla_\theta J(\pi_\theta)$ with respect to π, based on its original definition.

If $P(\tau|\theta)$ denotes the probability of trajectory τ under policy parameters θ then one can show that:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim T,\pi}\left[R(\tau)\right] = \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) = \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) \\ &= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) = \mathbb{E}_{\tau\sim T,\pi}\left[\nabla_\theta \log P(\tau|\theta)R(\tau)\right],\end{aligned}$$

where the logarithm derivative expression:

$$\nabla_\theta \log P(\tau|\theta) = \frac{1}{P(\tau|\theta)}\nabla_\theta P(\tau|\theta)$$

and the inversion between gradient and integral enable to express the gradient of an expectation as the expectation of a gradient. The probability of trajectory τ is:

$$P(\tau|\theta) = P(s_0)\Pi_{t=0}^{N-1} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

where N is the length of τ. Then the gradient of the log-probability of trajectory τ is simply:

$$\nabla_\theta \log P(\tau|\theta) = \sum_{t=0}^{N+1} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

since only π_θ depends on θ. Then, injecting this expression in the policy gradient one can deduce the "Expected grad-log-prob" expression:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim T,\pi}\left[\sum_{t=0}^{N+1} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right]. \tag{2.10}$$

This way, one can easily build an estimator using the collected trajectories.

The second issue concerns the need for unbiased and low-variance sample estimates of the policy gradient. It appears that direct estimators are unbiased but contain uncorrelated terms that needlessly increase the variance. To converge such an estimator one then needs to gather a larger amount of samples. Thus, sample-efficient training methods should rely on different low-variance, surrogate estimators. Looking at equation 2.10, the first simple modification reducing variance, concerns the replacement of the return $R(\tau)$ by its *reward-to-go* version. For a given time t, one can split $R(\tau)$ into the partial sum of rewards obtained before t (in its past) and the sum of rewards obtained after t (in its future). It is common intuition that the sampling probability $\pi(a_t|s_t)$ of action a_t at time t should only be optimized considering the future rewards (i.e. the consequences of the control action). The first partial sum is then a yet unbiased but uncorrelated source of noise for the estimators built on this definition:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim T,\pi}\left[\sum_{t=0}^{N+1} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\cancel{\sum_{t'=0}^{t-1} \gamma^{t'} r(s_{t'}, a_{t'}, s_{t'+1})} + \sum_{t'=t}^{N-1} \gamma^{t'} r(s_{t'}, a_{t'}, s_{t'+1})\right)\right].$$

The fact that $\mathbb{E}_{\tau\sim T,\pi}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) f(s_t)\right] = 0$ for any function f that only depends on the state[8], enables to subtract to the reward-to-go, functions called *baselines* that help reduce the variance of the estimator as analyzed by Greensmith *et al.* [81]. Choosing the estimated value function V^π as baseline or using the previously introduced action-value function Q^π to replace the reward-to-go are both valid choices. Combining both leads to the very common baseline choice of the advantage function A^π:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t),$$

which quantifies the "advantage" of taking action a_t instead of the best action according to policy π, all things kept equal. This enables to disentangle the effect of a_t yielding consequences through the immediate reward and the current state transition only (according to the Markov property), from the following actions' effects, with the effect of reducing estimator variance.

The value of the discount factor γ plays an important role in the variance of these estimators. Large γ values (close to 1) enable to learn more long-term policies that perform better but at the cost of larger estimator variances, whereas small values favor nearsighted and greedy policies and small variances. Thus, for undiscounted MDPs ($\gamma = 1$), the constraint of low estimator variance often constrains to choose lower values of γ, thus introducing a bias between the optimization objective

[8] This can be proven using $\nabla_\theta f(s_t) = \nabla_\theta \int_{a_t} \log \pi_\theta(a_t|s_t) f(s_t) = 0$.

and its estimator. To mitigate this bias, Schulman *et al.* [211] propose Generalized Advantage Estimation (GAE), an hybrid unbiased advantage estimator (considering $\gamma = 1$) making use of the self-consistency of the Temporal Difference (TD) residual, to propose an efficient trade-off between the perceived bias on γ and the variance of the estimator. Even though conceived for undiscounted problems, GAE is the advantage estimation method in most of the state-of-the-art training algorithms.

2.2.3.4 Value bootstrapping

As explained earlier, policy-iteration algorithms use the sample sequence collected during roll-outs to directly estimate the return of each state and the corresponding advantage. Yet, roll-outs must have a maximum length even when not imposed by the environment. This stopping constraint, may introduce a bias on the return and all other estimators computed with it, since the last terms of the discounted sum of rewards are not collected. As this "artificial" cut-off of the trajectory should obviously be transparent for the optimization process, this final reward is usually replaced by the best estimate of the expected sum of rewards, i.e. the value function V evaluated on the last state s of the trajectory. Thus, contrary to other non-terminal states, one "bootstraps" this state value in a TD fashion similarly to value-iteration algorithms.

2.2.3.5 Hybridization

Value- and policy-iteration methods not being incompatible, numerous hybrid algorithms have been proposed, taking advantage of the best of each paradigm. One can cite DDPG [143], TD3 [73] or SAC [88]. As seen before, model-free algorithms are approaching methods solving practical implementation issues and thus making trade-offs between mathematical tractability and computational cost on the one hand, and accuracy and failure modes on the other hand. Contrary to the model-free/model-based split, the distinction between Value- and Policy-iteration is not mutually exclusive so it should be thought of as a continuous spectrum of methods proposing a range of trade-offs between the advantages and drawbacks of both paradigms.

Bucci *et al.* [38] as well as Zeng & Graham [266] used DDPG to drive the One-dimensional Kuramoto-Sivashinsky equation (refer to sections 3.4.1 and 4.1) toward each of its non-trivial fixed points. Koizumi *et al.* [126] also implemented this algorithm on a cylinder flow.

2.2.4 Model-based RL

2.2.4.1 A trade-off between cost and accuracy

Despite the multiplicity of proposed methods mitigating the issue of sample efficiency for model-free RL (such as off-policy methods and/or experience replay), this family of training algorithm suffers, especially in the case of costly environments, from its low exploitation of the collected data. From that view-point, model-based approaches are appealing thanks to their promise of learning the internal dynamics of the environment and thus, as said by Moerland *et al.* [158], "the model is a form of reversible access to the MDP dynamics". This way, states and transitions can be replayed at a much lower cost than on the environment in order to train the agent. This "reversible access" is all the more important in the case of fluid mechanics where replaying any arbitrary partial state is impossible or very hard due to the large number of degrees of freedom of the system. Experimentally it is impossible (unless by luck of reaching this state on a trajectory) and numerically it would mean storing an impractical amount of full-state snapshots of the flow.

Yet, (reduced-)model accuracy often appears as a limitation to the agent's performances. As the querying horizon (length of the model roll-outs) increases, approximation errors accumulate, raising the issue of accurate multi-step prediction. Thus, a crucial trade-off between model-simulated error-prone cheap data and real-world expensive and accurate data must be found. This trade-off obviously depends on the complexity (linearity, dimensionality and observability) of the dynamics of the environment. Improvement guarantees, in the form of a lower-bound have been considered by Luo *et al.* [147]. Janner *et al.* [105] also discussed this issue and leverage the accuracy of the model to tune the roll-out horizon on their model.

2.2.4.2 Modeling the reduced-order dynamics

In the extreme case where the dynamics is known without error such as for board games [209], the whole training process may be run on the model, since its accuracy its guaranteed. In most of the cases however, the prior knowledge about the system is either nonexistent or partial. In the first case, linear methods [137, 130] such as the ones introduced in section 2.1.5.3, Gaussian processes [62], or Gaussian Mixture Models can be used to model the dynamics and its associated uncertainty. Though, thanks to their interpolation capabilities even in large dimensions, NNs (refer to section 2.3.2 for more details) are often chosen for their high sample efficiency. Gal *et al.* [74] for instance re-used the PILCO algorithm introduced by Deisenroth & Rasmussen [62] but resorted to Bayesian NNs instead of Gaussian processes. Kaiser *et al.* [117] used an auto-encoder structure on video-game snapshots to predict the evolution of the system and reduce data collection needs on the real environment while still improving the learning speed. Chua *et al.* [53] introduced Probabilistic Ensembles with Trajectory Sampling (PETS), a learning method that leverages the disagreement between multiple NN-based models to assert the average model accuracy. Using Model Predictive Control (MPC) they derive an implicitly-defined policy based on the learned dynamics.

When prior knowledge on the dynamics or at least its structure can be assumed, restricting the model with these constraints may help improve its accuracy, especially for multi-step predictions where stability and/or prevention of distribution collapse is crucial. The linearity assumption can lead to models such as the already-mentioned DMD with control (DMDc) or Koopman decompositions. Assuming that the dynamics project well on a linear combination of observable functions leads to approaches like SINDY with control [116], that can be considered as model-based RL.

Chen *et al.* [48] proposed neural ODEs, using a time-continuous approach, instead of the traditional discrete time ($x_{t+1} = x_t + f(x_t, t, \theta)$) approach. Yin *et al.* [264] introduced APHYNITY in order to "leverage prior dynamical ODE/PDE knowledge in situations where this physical model is incomplete, i.e. unable to represent the whole complexity of observed data". They proposed a bi-headed reduced-order model, where a tunable PDE is complemented with a data-driven NN similar to a neural ODE. Greydanus *et al.* [82] introduced Hamiltonian Neural Networks (HNN) aiming at guaranteeing energy conservation. Chen *et al.* [50] proposed a recurrent NN stepped forward in time so that it conserves Hamiltonian quantities. Other quantities such as mass conservation or Total Variation Diminishing properties (TVD) may also be tackled by the family of Physics-Informed Neural Networks [40] (PINN).

2.2.5 A quick focus on PPO-CMA: Balancing exploration and chaos

Proximal Policy Optimization with Covariance Matrix Adaptation (PPO-CMA) was proposed by Hämäläinen *et al.* [89] as a variant of PPO, aiming at avoiding the "premature shrink [in] the exploration variance". As opposed to PPO, where the exploration variance σ ($a \sim \pi_\theta(s) = \mathcal{N}(\mu_\theta, \sigma)$)

is a user-defined parameter (constant or scheduled), PPO-CMA relies on CMA-ES-like methods to output a data-adapted σ vector from the policy π as shown by figure 2.3. Both outputs μ and σ of the policy are trained with surrogate losses using GAE (like PPO) but that differ from the clipped formulation proposed by Schulman *et al.* [212]. The loss on σ (trained before μ) is:

$$\mathcal{L}_\sigma = -\frac{1}{\#\mathcal{H}} \sum_{t=1}^{\#\mathcal{H}} \text{ReLU}\left(A^\pi(s_t, a_t)\right) \log \pi_\theta(a_t|s_t),$$

where $\mathcal{H}$ is a sub-batch of collected data from the last epochs $\#\mathcal{H}$ its cardinal and ReLU is the rectified linear unit activation function (i.e. $\max(\cdot, 0)$). Here negative advantage samples are totally discarded concerning the loss on σ. The formulation of the loss on μ proposes mirroring mechanisms that uses these negative advantage actions to "push" the actor in the opposite direction:

$$\bar{a}_t = 2\mu_\theta(s_t) - a_t \quad \text{mirrored action}$$

$$\kappa_t = \sum_j e^{-\frac{\left((a_{t,j} - \mu_{\theta,j}(s_t)\right)^2}{2\alpha\sigma_{\theta,j}(s_t)^2}} \quad \text{avoidance kernel}$$

$$\mathcal{L}_\mu = -\frac{1}{\#\mathcal{B}} \sum_{t=1}^{\#\mathcal{B}} \left[\text{ReLU}\left(A^\pi(s_t, a_t)\right) \log \pi_\theta(a_t|s_t) - \text{ReLU}\left(-A^\pi(s_t, a_t)\right) \kappa_t \log \pi_\theta(\bar{a}_t|s_t)\right],$$

where α is a predefined damping parameter that drives the "avoidance kernel", tasked with favoring small amplitude negative advantages (likely to be less destabilizing) over larger negative ones. $\mathcal{B}$ is the collected data batch of the current epoch only and $\#\mathcal{B}$ its cardinal.

This way σ is automatically tuned, avoiding the tedious search on the appropriate value or schedule required by PPO. In flow control cases, nearly chaotic dynamics require a precise tuning of the exploration noise. In order to prevent complete divergence of the observed states due to the sampling noise but with the aim to maintain the ability to explore new regions of the state-action space, PPO-CMA mechanisms appear particularly suited. In the implementation of the current study, the output on σ is fitted with a manually tunable variable that enables to set the initial order of magnitude of σ based on the input observations on the first training epoch.

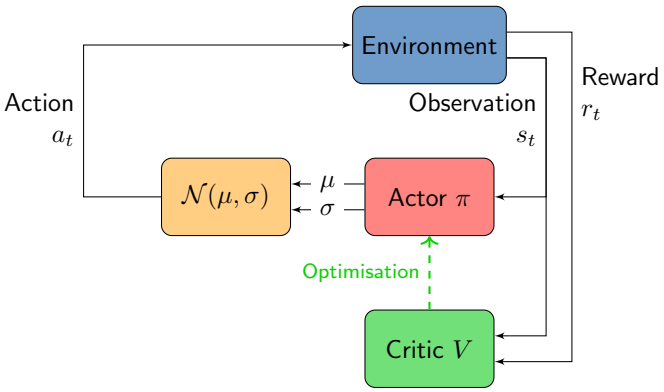


Figure 2.3: PPO-CMA agent structure

2.2.6 A quick focus on PETS-MPC: A flexible, implicitly defined policy

As introduced earlier, Probabilistic Ensemble Trajectory Sampling for Model Predictive Control (PETS-MPC) has been proposed by Chua *et al.* [53]. It leverages two interesting principles: the

first concerns ensemble averaging and the second is about Model Predictive Control (MPC), namely considering the control policy as the result of a local optimization of the control actions over a few steps in the future. This way, the policy is implicitly defined and thus requires extra processing compared to a simple forward pass in a NN but in turn allows for adding extra constraints that can be dynamically implemented in the optimization process (such as action clipping). As shown by figure 2.4, multiple models, each embodied by a dense NN, get the current observation s_t and control action a_t as input, are tasked with estimating the best Gaussian distribution $s_{t+1} \sim \mathcal{N}(\mu(s_t, a_t), \sigma(s_t, a_t)$ and are trained on data samples gathered from roll-outs on the full state environment. The idea behind having multiple models that are supposed to display rigorously the same behavior is linked to **epistemic and aleatory uncertainties**.

2.2.6.1 Epistemic and aleatory uncertainties

Aleatory uncertainty concerns the perceived yet intrinsic randomness of the dynamics of an environment. On fully-observed systems, aleatory uncertainty directly relates to the stochasticity of the dynamics. Thus, a fully-observed deterministic environment has no aleatory uncertainty. But if it is partially observed (which generally is the case for fluid mechanics), despite behaving deterministically, it may show aleatory uncertainty in the sense that starting from the same partial state (but potentially different full-state) and stepping the environment forward may lead to different partial states, and this has nothing to do with chaoticity. Partially observed environments present aleatory uncertainty simply because of non-observed or "hidden" informative variables. A large aleatory uncertainty might jeopardize training, since it would lead the agent to consider two widely different full-states as identical. Thus locating observations properly is part of what makes the expertise in aerodynamics, in order to capture the phenomena relevant to the control policy and possibly drop less meaningful ones. Aleatory uncertainty is thus strongly related to the notions of observability and detectability introduced in section 2.1.5.2.

Conversely epistemic uncertainty relates to the current knowledge gathered by an agent about the dynamics of the environment, or in other words, to the confidence in the predictions an agent can make about the environment given its experience. Two agents (accessing the same observations) may have a different epistemic uncertainty if one is better at learning than the other but will be confronted to the same aleatory uncertainty. The epistemic uncertainty can be reduced by training the agent on more sample data or by improving the training method. A perfectly trained model would have a null epistemic uncertainty, but as these two uncertainties add-up, its prediction will remain somehow uncertain if the system shows aleatory uncertainty.

Important note: These current definitions may differ from these used in other domains of ML or statistics, especially concerning partial observation which is usually considered as a source of epistemic uncertainty. But here, as the observation layout is a fixed parameter of the problem and since the environment is considered as black-box, it is more logical to consider it as aleatory uncertainty.

2.2.6.2 Training and action inference

Here, having multiple models helps separating these two types of uncertainty. For a given prediction of a partial-state trajectory, the disagreement between the models gives an estimate of the epistemic uncertainty and the own uncertainty (σ) of each model prediction aims at inferring aleatory uncertainty. Running an ensemble average over all the model predictions ensures more precise and

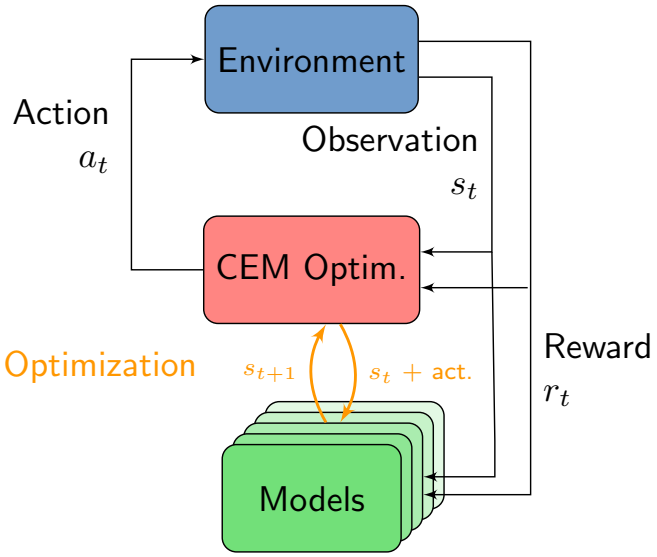


Figure 2.4: PETS-MPC agent structure

more stable predicted trajectories. To ensure that all models do not collapse on the same behavior because of overfitting, they are trained with different data samples.

Concerning action inference, the policy being implicit, each action query triggers an optimization. To do so, the implementation proposed by Chua *et al.* [53] relies on the Cross-Entropy-Method proposed by Botev *et al.* [29]. For a given prediction horizon h, a flock of action sequences are either generated randomly or recovered from the previous optimization. Each action sequence is evaluated on the models multiple times, using "particles" initialized with the current observation s_t and stepped forward in time by the models on h steps. Once all the predicted trajectories are synthesized, their cost can be computed. The average cost of a given action sequence is used to rank the sequences and re-sample better action sequences. At the end of the optimization the first n actions of the best action sequence are kept to be implemented on the real environment (generally $n = 1$ and this optimization thus occurs at every control step).

The original method assumes that the reward is a (known) function of the observations only. Yet, on more complex environments this is generally not the case. Thus the current implementation augments the observation vector s_t with the current reward r_t, which then becomes yet another quantity to estimate, potentially reducing the performances of the method. The current implementation also uses a residual connection, hence in practice the NN of each model outputs $\Delta_s = s_{t+1} - s_t$, which is similar to a discrete derivative of the signal.

2.2.7 Policy distillation

Policy distillation was first introduced by Rusu *et al.* [200] as a way to transfer policies from a trained agent to another untrained agent. This may be used as a reduction method, by distilling from a large network to a smaller one (empirically harder to train) or to combine multiple single-task policies into one single multi-task expert policy. Contrary to most forms of transfer learning that consider direct networks' parameters (weights and biases) transfer, policy distillation works by imitation of a "teacher" agent, containing the expert pre-trained policy, by a "student" one, hosting the un-trained policy. This approach was successfully used by Teh *et al.* [238] and later by Lai *et al.* [132] who proposed an extension called Dual Policy Distillation. This method implements a

bidirectional knowledge transfer. In that case both policies are alternatively teacher and student, they "extract beneficial knowledge from each other to help their learning" as explained by the authors. This approach resonates with the previously introduced idea of ensemble expertise where a group of trained neural structures performs generally better than each of its members taken individually.

2.3 Neural networks

Non-linear control and reinforcement learning express the need for a versatile and easily optimizable object capable of embodying the control laws (policies) and other stochastic estimators. Neural Networks (NNs) are a biologically inspired and trending solution to this issue. The idea behind these computing systems dates back to the 1940s and efforts of neuroscientists to understand the brain structure and operation. Artificial neural networks implement this same idea of thresholded message propagation as a way to perform computations. While the *perceptron* developped by Rosenblatt [194] can be considered as the first implemented artificial neural network, and that most of the theoretical ground work has been made in the 1970s, 1980s and 1990s [104, 248, 208, 77, 98], one had to wait until the years 2000s to see a widen use of neural network in numerous scientific domains, mostly thanks to a sustained and uninterrupted growth of computational capacities from the 1960s onward, as theorized by the famous Moore's law. The usage of NNs has since then been exploding in a wide variety of scientific domains such as image analysis [142], data-mining [167], natural language processing [171], robotics and autonomous vehicle control [131], cybersecurity [23], chemistry and drug discovery [46], finance [172] or physics [41].

2.3.1 A layered structure

In the following, the most general Fully Connected Neural Network (FCNN) architecture is described. Most of the other architectures re-use (partially or completely) the ideas behind this genuine structure. Thus by default, the "NN" abbreviation will refer to a FCNN in the following. A FCNN consists in a series of fully connected layers that cascade into one another. Each layer L transforms a given input vector $x \in \mathrm{R}^n$ into an output vector $y \in \mathrm{R}^m$ and consists in a matrix product followed by a generally non-linear activation:

$$y = L(x) = f\left(Wx + b\right),$$

where $W \in \mathcal{L}(\mathrm{R}^n, \mathrm{R}^m)$ is a weight matrix, $b \in \mathrm{R}^m$ is a bias vector and f is a (scalar) activation function. Starting from an input vector x, the output y_{NN} of a (Deep) FCNN is thus simply computed in a forward pass on the network as:

$$\begin{aligned} y^0 = x \quad \text{and} \quad y^{l+1} = L^l(y^l) \quad \forall\, l \in \{0, \dots, n-1\} \\ y^n = L^{n-1} \circ L^{n-2} \dots L^1 \circ L^0(x), \end{aligned} \tag{2.11}$$

by composition of the n layers of the NN. The activation function aims at imposing a non-linearity or a filtering threshold. Rectified linear unit (ReLU) and logistic (sigmoid) functions are the two most used activations. Together composed with the matrix product and the bias, the layer function L can be considered as a "ridge" function for each of its output components, extracting m scalar features of the input x. The commonly used representation of neural networks layers as a group of individuals "neurons" comes from this consideration of a layer as feature extractor, each neuron

being in charge of collecting the signal output from upstream layers (or the input) and computing a given feature signal. On a FCNN with more than one layer (also called multilayer perceptron), the last layer is defined as the "output layer" and it dimension m is constrained by the dimension of the output space, while all the other layers are called "hidden layers", their dimensions not being constrained (except to first one that matches to one of the input space).

2.3.2 A universal approximator

The universal approximation theorem mathematically ensures that a multilayer perceptron (a FCNN with at least one hidden layer) can approximate any continuous function $g : \mathrm{R}^n \mapsto \mathrm{R}^m$ to an arbitrary precision as long as the activation function is not polynomial and the number of extracted features is large enough. In other words a large-enough linear combination of feature extractors (as defined above) can uniformly converge toward any continuous function.

Yet, this property and the convergence bounds it provides are useless and unpractical since the number of parameters and maximum computational cost it guarantees become quickly beyond reach when the input space dimensionality increases and/or the regularity of the ground-truth function g (in a Sobolev sense) drops. In practice, accurate function approximation by relatively "small"[9] neural networks is observed. The fact that NNs seem to break this "curse of dimensionality" has been explained by Barron's theorem [12], proving that, for slightly different regularity hypotheses, the upper bound of the approximation error could be drastically diminished and would not depend on the inputs dimension. Yet, this does not entirely explain the efficiency of NNs at approximating functions. The roots of the proficiency of NNs come from the fact that the functions f quantify characteristics of an underlying physical phenomenon which is itself driven by smooth and "structured" laws. Second, the approximation of g is only required for a very reduced portion $\mathcal{S} \subset \mathcal{R}^n$ of the input space (where the training inputs samples lie). Thus the problem at hand is not to get a general-purpose approximation of a function g on the whole input space, as computed by traditional (non)-linear decomposition methods, but rather to model the relation $y = g(x)$ by a decomposition on data-adapted bases (trained ridge functions) on reduced supports of the input space.

2.3.3 Training a neural network

2.3.3.1 Gradient back-propagation and related issues

The accuracy of a NN approximating $y = g(x)$ is generally measured by a scalar metric of the approximation error on the input data $\mathcal{S}$ called the "loss" or the "cost function" (here denoted $C : \mathcal{R}^n \mapsto \mathcal{R}$). Training a NN thus corresponds to modifying its parameters θ (matrices of weights W_i and biases b_i) in order to minimize the loss function over the training data-set:

$$\theta^* = \arg\min_{\theta} ||C||_{\mathcal{S}}. \tag{2.12}$$

where θ^* is the optimized value (overall best) of θ and $||C||_{\mathcal{S}}$ a measure of C over the input data. Back-propagation [248] by automatic differentiation is a cornerstone enabler of the approximation power of neural networks, since it enables this optimization of the parameters at a reasonable computational cost. It consists in computing the gradient of the loss function C with respect to

[9] whose size is orders of magnitudes smaller than the theoretical bound

each parameter $\theta = \{W^0, b^0, \dots, W^{n-1}, b^{n-1}\}$ of the NNs, using the composing formula of equation 2.11. Let us define δ_j^l as:

$$\delta_i^l \equiv \frac{\partial C}{\partial y_i^l},$$

which can be considered as the sensitivity of C to the j^{th} neuron activated output signal of layer l: $y_i^l = L^{l-1}(y^{l-1})_i$. If $l < n$, the right-hand side can be re-written as:

$$\frac{\partial C}{\partial y_j^l} = \sum_i \frac{\partial C}{\partial y_i^{l+1}} \frac{\partial y_i^{l+1}}{\partial y_j^l}$$

$$\text{as} \quad y_i^{l+1} = f\left(\sum_j W_{ij}^l y_j^l + b_i^l\right), \quad \text{then} \quad \frac{\partial y_i^{l+1}}{\partial y_j^l} = f'\underbrace{\left(\sum_j W_{ij}^l y_j^l + b_i^l\right)}_{z_i^l} W_{ij}^l$$

$$\text{and thus} \quad \delta_j^l = \sum_i \delta_i^{l+1} f'\left(z_i^l\right) W_{ij}^l$$

$$\text{or} \quad \delta^l = \left[\left(W^l\right)^T \delta^{i+1}\right] \odot f'(z^l) \quad \forall\, l \in \{0, \dots, n-1\},$$

where $\odot$ is the Hadamard product. The value of these δ^l can thus be computed recurrently, starting from δ^n, from the last layer to the first. One can finally deduce the gradient of C with respect to biases b_i^l and weights W_{ij}^l as:

$$\frac{\partial C}{\partial b_i^l} = \delta_i^l f'(z_i^l) \tag{2.13}$$

$$\frac{\partial C}{\partial W_{ij}^l} = \delta_i^l f'(z_i^l) y_j^{l-1} \tag{2.14}$$

Thus, to minimize $||C||_{\mathcal{S}}$ over the training data, one must update the parameters θ of the neural network in a direction opposite to $\nabla_\theta ||C||_{\mathcal{S}}$ so that:

$$\theta \leftarrow \theta - \alpha \nabla_\theta ||C||_{\mathcal{S}},$$

where α is an update parameter called the "learning rate". α can either be a predefined constant or be computed algorithmically in order to increase convergence speed and/or accuracy. Adam [124], the best known optimizer for instance, bases the computation of α on the L_2-norm of $\nabla_\theta ||C||_{\mathcal{S}}$ in order to make larger updates in regions where the gradient is low. This parameter update guarantees a decrease of $||C||_{\mathcal{S}}$. Yet, as the neural network embodies a non-linear function, this procedure may end-up in a local minimum. Secondly, the computation of $||C||_{\mathcal{S}}$ may be expensive and an estimate of this measure over a reduced portion of the input data, called a "mini-batch" $\mathcal{B} \subset \mathcal{S}$ may be a relevant approximation. This cheaper estimator provides less accurate update directions for θ but this error may avoid local minima or enable to leave them. This is the idea behind the Stochastic Gradient Descent (SGD). Most of the time, the measure $||C||_{\mathcal{B}}$ is a simple mean value computation:

$$||C||_{\mathcal{B}} = \frac{1}{\#\mathcal{B}} \sum_{x \in \mathcal{B}} C(x),$$

$\#\mathcal{B}$ being the cardinal of mini-batch $\mathcal{B}$.

Gradients of the loss C with respect to network's parameters θ involve a product of multiple first-order derivatives of the activation function f', especially for many-layered networks. Two spurious, different behaviors may arise. If $|f'|$ has values larger than one, the product of these may shoot-up and provide huge gradient values especially on the upstream-most layers. Combined with the gradient estimation errors due to "mini-batching", this could end up in a catastrophic parameter update. To avoid the so-called "exploding gradient problem", it is recommended to use activation functions which gradient is always in $[-1, 1]$. On the other hand and as highlighted by Hochreiter *et al.* [98], the product of derivative may be vanishingly small, preventing parameter update and thus possibly freezing training. Rectifiers (such as ReLU) suffer less from this flaw since they only saturate in one direction. This "vanishing gradient problem" can also be tackled by a change in the network architecture as described below.

2.3.3.2 Miscellaneous commonly implemented practices

Even the training of a simple FCNN requires a significant number of hyper-parameters to be completely described and set up. This section provides details about both these implementation details and the rules of thumb used to set these hyper-parameters.

Initialization of the neural networks' weights is to be carefully thought in order to both prevent an explosion of the first output values and to enable successful updates. Uniform initialization is not suited since it does not "break the symmetry" between the neurons and thus leads to uniform gradients, hugely limiting the interest of wide neural layers. An uneven initialization is then required in order to unleash the approximation power of the NN. This is generally done using a random initialization. Yet depending on the type of activation functions, vanishing-gradient issues may arise. To alleviate this, specific initialization procedures, that adapt to the activation function have been proposed, such as the "Xavier" initialization [80] for sigmoid activations or the "He" initialization [91] for rectifiers.

In supervised learning, over-fitting is a spurious phenomenon where the neural network not only learns the main patterns of the input-output relation but also the noise and/or the biases due to sampling, thus solving the bias-variance trade-off by a low bias and high variance solution to over-fit all the samples. This generally occurs when the training has been led for too many epochs and causes poor generalization performances on the test data. Sanity checks such as comparing both validation (average loss on the training data) and test (average loss on unseen data) losses help identify this issue. Drop-out is another regularizing trick, that slightly lengthens training costs for a given network but tends to prevent over-fitting. The basic idea is to ignore the signal provided by a fraction of randomly selected neurons. The learning of inter-dependency between neurons is thus reduced and an increased robustness is empirically observed. Other methods such as the weight decay where a L_1 or L_2 penalization on the weights is added to the training loss, or batch normalization where the input data is applied (refer to 7.3 for more details) are also commonly used to reduce over-fitting. All these methods rely on hyper-parameters (dropout rate, batch sizes, or penalization coefficients) that are generally tuned empirically.

NN's number of hidden layer and hidden widths is also a topic for which choices are often made based on past experience and empirical tests. Some rule-of-thumb sizing methods have been proposed but these rely more on these observations rather that mathematical evidence. For known analytic functions $y = f(x)$, where the minimum architecture can be computed, it is observed that efficient training is only achieved for neural networks much larger than this lower bound. This may come from the previously introduced issues of symmetry breaking between neurons of the same layer

and poor conditioning causing vanishing gradients. Overall, smaller networks tend to learn slower than larger ones, as shown by figure 2.5. Thus, as long as optimizing costs are not limiting training, largely over-sized neural networks are chosen, despite the expected complexity of the function to fit. Similarly, the optimizer's learning rate (or more accurately the parameter driving the computation learning rate for adaptive gradient descent optimizers such as Adam) is set based on the optimal values found on similar cases.

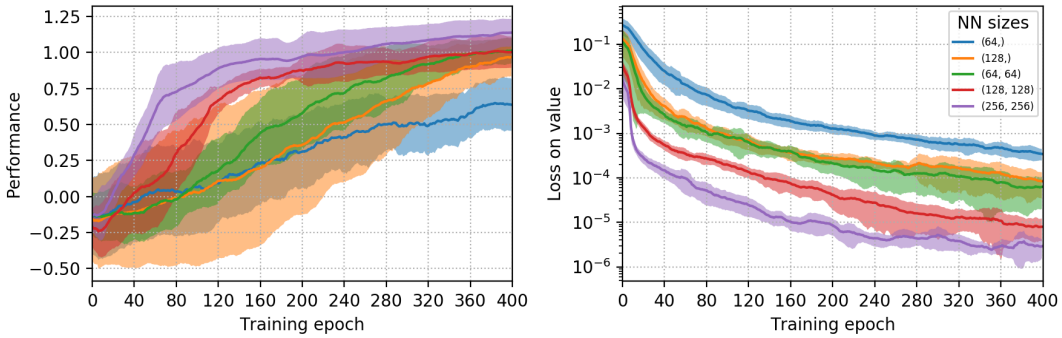


Figure 2.5: Effect of the architecture of neural networks on both the performance and the loss on the value function of a control cylinder flow.

Pruning consists in permanently deleting neurons after training, in order to reduce its size in memory and cost for a forward pass, while still preserving performance. Pruning can be performed in a structured fashion (e.g. filter pruning for convolutional NN) with the idea of entirely removing channels or layers, or in an unstructured fashion, where based on a pruning criterion (norm, sign, ...), weights and biases are set to zero. In the latter case, gains are only observed if underlying ML framework supports sparse matrix computation and storage. Thus, except in the context of MLOps, pruning is rarely implemented for prototype systems.

2.3.4 Other architectures

All the issues discussed until here concern the "basic" FCNN architecture, but these extend also to other types of NN, such as the ones (non-exhaustive list) discussed in this part. Convolutional Neural Networks (CNN or ConvNets) are mostly used in image processing. Their layers do not implement matrix multiplication followed by an activation but translation-equivariant convolutions of predefined stride and kernel width, possibly followed by an activation. Parameters are the kernel matrices that convolve and combine the different channels output by the upstream layer. This architecture makes it possible to process large input sizes while keeping a moderate number of parameters (compared to an equivalent FCNN) thanks to weight-sharing (one single kernel is need for a convolution over the whole input).

Recurrent Neural Network (RNN) is a class of neural networks, particularly suited for processing data sequences, that can reuse previous outputs as input, while also having hidden states. This architecture allows to take the causal links between prior and current samples into account and can be seen as "machines" that learn how to step a given dynamics forward in time. These neural networks are known to be harder to train that other architectures since they have a large equivalent layer depth that amplifies the issues of vanishing/exploding gradients. The two best known RNN architectures are the Gated Recurrent Unit [51] (GRU) and the Long-Short Term Memory Unit

[99, 77] (LSTM), the latter being an evolution of the traditional RNNs, dealing with these gradient issues.

Another solution to the vanishing/exploding gradient problem stemming from having to propagate signal across a large number of layers is the Residual Neural Network [232] (ResNet). Is contains "skip" connections that shortcut some layers. This imposes a constraint on the sizes of these skip-connected layers but also enables a much easier gradient back-propagation trough these jump-over connections, that act as "highways".

The following architectures are more to be considered as an assembling of neural networks basic bricks rather than genuine flavors of NNs. These meta-architectures are generally composed of differently trained NNs that interact with each other. Autoencoders (AE) and their variational evolution (VAE) are used to both learn a dimensionally reduced representation of the input data and for their generative capabilities. The idea is first to encode an input into a small output vector (called the latent vector) and then to decode it back with the goal of matching the input. The training can thus be considered unsupervised. This can be used as a way to "denoise" data or detect anomalies, and to generate synthetic data using the decoder. This architecture shares strong links with traditional linear model reduction methods. A one-hidden-layer, linearly-activated auto-encoder will converge to the same sub-space as a Proper Orthogonal Decomposition (POD) of the same dimension (excepted that latent components are generally not orthognal). This approach was used by Kashima [120] as an efficient and denoising model reduction method.

Generative Adversarial Networks (GAN) are also an architecture used to learn how to generate outputs in a unsupervised manner. The idea is to indirectly train a generator network against a discriminator network. The latter is tasked with telling how much an input is "real" (genuine training data) or "synthetic" (output of the generator) and the generator is tasked to fool the discriminator. The training of such an architecture may fail if there is a large unbalance in the performances of both the generator and discriminator or may suffer from "mode collapse" when the generator ends up outputting the same data, taking advantage of poor discriminator convergence.

Capsule Neural Networks (CapsNets) have been proposed by [201] to both solve the issues of translational/rotational invariance of objects, spatial relationships between different entities and modeling hierarchical links. Relying in capsule units tasked with pose estimation and a forward mechanism of "routing by agreement", this architecture enables to better classify images.

At last and to tackle the issue of long-term dependencies in data sequences, especially in natural language processing (NLP), Transformer networks have been proposed. They use the principle of self-attention and context embedding to perform tasks on sequential data, such as machine translation [33], image synthesis or deepfake detection. These architectures are generally large and expensive to train, especially for long input sequences. Alternatives such as the Reformer [125] have been proposed to alleviate this issue.

2.4 Genetic programming

Genetic algorithms [100, 121] (GA) is a family of methods inspired by biological evolution. These methods aim at avoiding local optima by leveraging population-based meta-heuristics. They are used to evolve a population of solutions toward better performance. Each individual of the population is generally represented by a binary code (analogous to its DNA). This encoding enables an evolution process directly inspired from gene mutations and cross-overs, where a Darwinian selective pressure favors advantageous operations with respect to least performing ones. Mutations take the form of random bit-flip-like operations on the code of individual whereas cross-overs see two

individuals exchanging parts of their genetic patrimony. The Darwinian evolution pressure is represented by a fitness function, on which every individual of the population is evaluated. Depending on this fitness values, individuals are ranked and they are more or less likely to either be kept as-is in the current population (elitism), be replaced with randomly generated ones (initialization) or undergo mutations or cross-overs.

2.4.1 Encoding a policy as a binary code

Genetic Programming (GP) is a family of genetic algorithms that aim at encoding a program, a function or an algorithm into a code that is compatible with the evolution process of genetic algorithms, in order to leverage its paradigm. Three families of program encoding can be distinguished. The first and most traditional way of encoding operations is with tree structures, where each tree node encodes an operation using its leaves as inputs. Linear Genetic Programming (LGP) represents the program or function as a sequence of operations. Cartesian genetic programming uses a graph representation of the program.

Most of the applications of GP to flow control rely on LGP. To encode the analytical function that embodies the policy, most of the studies resort to Reverse Polish Notation (RPN). The idea is to store constants, inputs and intermediate results into numbered "memory slots". Algebraic operations are also encoded using a standard mapping or hashing function. Each atomic operation is represented by the identifiers of the memory slots corresponding to its inputs, then by the code of the applied operation and finally by the identifier of the memory slot where the output of the operation is stored. The full policy is thus represented by the sequence of the codes of its atomic operations, that is generally embodied by a matrix (each row being the code of an operation).

Similarly to biology all the genome of an individual is not useful to the expression of the control law. For instance, memory slots that are not considered as outputs may contain intermediate results that are not reused. These are called "introns". This phenomenon yields two consequences. First, two individuals may embody the same policy but with a different genome. Second, introns can be seen as latent properties/functions of the individual, that can be later activated by a mutation or a cross-over. Multiple strategies are then implemented in the literature to either avoid introns, eliminate individuals only differing by their introns, or conversely promote the expression of introns.

Duriez *et al.* [68] assessed the potential of genetically discovered closed-loop control laws on three different experimental test-cases. These genetic methods were also used by Debien *et al.* [61] to learn efficient control laws reducing near-wall fluctuations on a sharp edge ramp flow with fluidic vortex generators mounted upstream of the edge. On an Ahmed body flow, actuated by blowing on the trailing edges, Li *et al.* [141] managed to reduce frequency cross-talk in the wake using LGP. This technique has been used for open-loop control as well, using a sinusoidal multi-frequency input signal to control a mixing layer [138, 139]. Gradient-enriched LGP [56] was proposed as a way to extract local gradient approximations, in order to better converge toward loss optima. This approach was notably implemented on the fluidic pinball test-case to reduce wake fluctuations, on a numerical setup by Maceda *et al.* [148] and experimentally by Raibaudo & Martinuzzi [187].

2.4.2 The curse of dimensionality

While RL is not immune to the increased difficulty of finding global optima of a search space when its dimension increases, this issue is all the more important with GP, since its approach is global, in

the sense that performance can only be guaranteed if the search space is "reasonably well" explored. RL exploits the local "slope" of the optimization loss to improve. Thus as long as the policy lies in the attractive basin of an optima and the optimization method is well-tuned, one is sure to reach this optima. Conversely, local gradients are not exploited by GP, thus the average distance between a given individual and the optima is strongly linked to the sampling method (guided by fitness-oriented selection methods) and to the dimensionality of the space the individual evolves in. This distance thus scales exponentially with the dimension. This is one expression of the well-known "curse of dimensionality" that prevents most of the proposed AI approaches from scaling-up to real-world applications.

Thus in the context of flow control, GP methods are bound either to embody models having a low number of inputs, outputs and parameters or to be assisted by other mechanisms in order to be successful.

2.4.3 The exploration-exploitation dilemma

GP proposes a way to solve the exploration-exploitation dilemma different than what RL does. Where RL relies on local, gradient-based optimization, GP proposes a global approach leveraging population meta-heuristics. Considering the exploration-exploitation dilemma, while GP being by nature more turned to exploration, and RL more toward exploitation, both of these paradigms can be tuned to balance both objectives.

Elitism and more generally Darwinian selection pressure can be increased in GP to favor the individuals in the vicinity of (possible) optima. Here the trade-off can be expressed as a balance between population diversity and fitness of the best individual. If the balance leans toward exploration, the risk is to never find any performing individual. On the contrary, one may expect premature convergence (that may not even be on a local optimum).

Exploration noise (and other analogous methods of exploration) can be dynamically tuned to encourage exploration or exploitation in RL. Large exploration noises may as well prevent to find a performing policy, all the more that here the effects of this randomness accumulate with time and may render the system chaotic. Conversely, low exploration noise may hinder the optimization and "freeze" the agent.

Chapter 3

Methods and tools

This chapter introduces both the tools that have been used or developed for the study and the main test-cases on which RL (and to a lesser extent GP) algorithms have been benchmarked. Technical details developed in parts 3.1, 3.2 and 3.3 are not prerequisites for the following chapters.

Contents

3.1 Pre-existing tools and frameworks

This section introduces the pre-existing tools chosen as building blocks of the present study and justifies these choices.

3.1.1 Python and main processing packages

The choice of programming language was driven by the need for efficient CFD solver co-processing, the existence of artificial intelligence Application Programming Interfaces (API) and the ease of prototyping (i.e. scripting). Thus, Python was a go-to choice for this kind of needs. On-top of an extensive standard library, the python community proposes a wide variety of open-source, external packages. This multi-paradigm language is dynamically-type, garbage-collected and interpreted which renders prototyping easy as well as enabling the development of a more complex code-base. Most of AI packages were initially written in Python. Thus, extensive documentation, mature APIs and a large community of users makes development straightforward.

Neural structures and ML operations are handled by the Tensorflow[1] library. This choice was made based on the better maturity of Tensorflow at the beginning the Ph.D. thesis compared to other libraries and by the fact that the first open-source RL-agent were released by OpenAI and were originally written using Tensorflow[1]. Similarly to other comparable solutions, Tensorflow provides automatic differentiation, that enables to seamlessly back-propagate training gradients and a wide variety of mathematical operators. Models are built as an operation graph and persistence functions offer to serialize and save their architecture and parameters.

The well-known numpy library has been chosen for non-ML data management. This package provides support for multi-dimensional array manipulation and mathematical computation. Other standard packages such as matplotlib, for graphics, or the in-house Cassiopee [19] package providing pre-, co- and post-processing functions for CGNS-standard fluid simulation cases have been used.

3.1.2 Parallelism interfaces

Parallelism, required for ensuring performance, is mainly handled by two different interfaces, intervening at two levels of granularity: inter- and intra-process communication. The Message Passing Interface (MPI), is the *de facto* message-passing standard for inter-process communication and synchronization. The MPI4py package provides handy Python bindings to call standard communication routines. Intel cluster nodes (used here) take advantage of the optimized Intel® MPI library, that implements these specifications[2]. Intra-process (shared-memory) communication is handled by the OpenMP API. When using both parallelism paradigms, one generally refers to hybrid computing. OpenMP thread creation and management is generally handled by C or C++ low-level code modules (such as LAPACK) and is thus transparent to the end-user most of the time.

[1]The use of any other Machine Learning API such as PyTorch would provide the same results, computing performances possibly varying depending on the algorithms' implementation.

[2]more specifically the open-source MPICH specifications

3.1.3 CFD solvers

The choice(s) concerning the CFD solver was primarily driven by the possibility of developing efficient co-processing routines, since controlling a flow simulation requires back and forth communication between the simulation and the agent. The possibility to directly alter the "under-the-hood" source code was also taken into account for this choice. FastS experimental solver [60] was thus preferred over other in-house solvers such as elsA or Cedre. FastS is a scalar finite-volume solver, written in Fortran and C++ for the low- and mid-level functions and wrapped in Python. It benefits from a strong integration with the Cassiopee API and co-processing is incomparably facilitated by the direct access granted to the computation tree in-between two numerical iterations, compared to other solvers requiring much heavier coupling scripts (elsA) or inter-process pipes (FreeFem++) to exchange information. This way, warm-up phases and some memory management functions (directly implemented in Python) have been altered in order to support the change in boundary conditions that may otherwise cause memory leaks. The use of FreeFem++ has also been envisioned, but strong obstacles such as (not exhaustive) the lack of sub-communicator splitting, deficient documentation and maintenance, doubts concerning scalability, package incompatibility and mainly its development on a specifically-built C++-derived language preventing from any integration of third-party packages, deterred from building RL-based flow control around it. FeniCs has also been investigated and first tests as well as draft of code interface architecture have been developed. Yet, all the studies and results presented in the current manuscript rely solely on FastS.

3.2 RLFramework

In order to properly interface CFD solvers and ML packages and reduce the quantity of code needed to fully develop a case and test it, multiple tools have been developed. They aim at increasing productivity by maximizing code reuse and reducing user-induced bugs. The Python framework introduced in this section is the backbone of all the studies that have been led. Emerging from the initial observation that these test cases share the same need for reliable, efficient and scalable co-processing between CFD simulation and RL training algorithms, this package has gradually been developed to incorporate an ever-increasing set of functionalities that enable an augmented productivity as shown by figure 3.1.

3.2.1 Paradigms

First, environments share a large number of characteristics together and learning agents do as well. Second, as this code is developed for research purposes, one should be able to incorporate new environments or learning algorithms seamlessly, and to switch from one environment to another without having to perform heavy adaptations of the implementation of the learning algorithm. This brings us to the notion of **modularity**. The idea of modularity develops the concept of interchangeable modules. Interchangeability implies two families of constraints: **functional completeness** or independence and **interface commonality**.

Functional completeness simply requires that each interchangeable module is self-sufficient, meaning that its state and internal dynamics are handled by functions or methods inside this module in a "black-box" fashion from the other modules. A module should also implement a limited and coherent set of functionalities that embody one logical structure to, for instance, enable easy unitary testing of each component. This means, for instance, that functionalities from RL

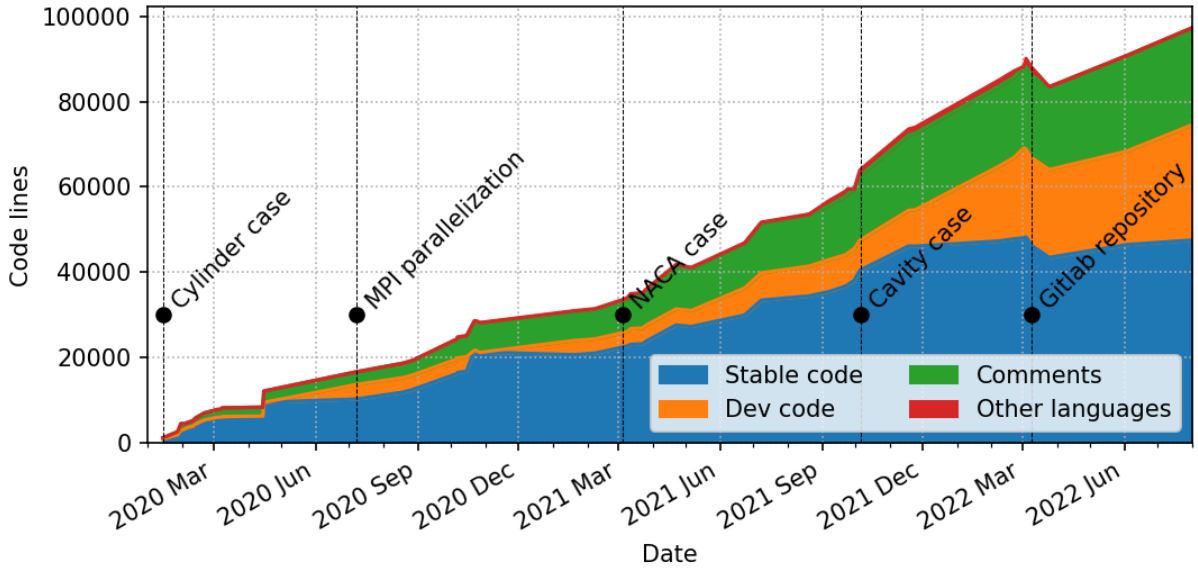


Figure 3.1: Evolution of the size of the code framework and event timeline.

agents won't be implemented in the same module as environment's internal dynamics methods. This need for structured and cloistered sets of functionalities makes class-based **Object-Oriented Programming** (OOP) relevant for the current framework. This programming paradigm considers classes, that define data models or organizations and provide the available procedures (or methods) that describe the internal dynamics of the defined type. Objects are instances of these classes, meaning that their type is the one of the class they are created from and that they can run the methods defined by their class. Classes can be seen as an assembly of definitions and laws, whereas objects implementing these classes are the corresponding real-world entities.

Considering a training case as one object with its own mechanics and logic, that contains multiple nested objects, that themselves implement parts of this logic and work with their own set of rules, enables an easier conceptualization of the code as a hierarchy of objects/agents interacting with each other. Independence and the need for interaction bring up the constraints related to interface commonality. These correspond to the fact that objects of the same family, for instance environments, must provide a fixed set of attributes and methods that are queryable by external objects. In order words, to enable seamless interchangeable modules, their interactions must be codified and all implementations of these modules (i.e. different class definitions) must comply with these interface contracts.

As Python does not originally enable interface declaration, these are defined via abstract class inheritance. Inheritance is also leveraged to boost code reuse. While different objects must be able to work independently from each other and only resort to interface-defined methods and attributes to exchange with other objects, their classes still share different degrees of similarity. For instance different environments, all complying with the "environment interface", but differing by the nature and dynamics of the environment they implement, may still share the same mechanics related to data buffering. In order to take advantage of these similarities, class inheritance is leveraged to implement a given function once and only once, and then to broadcast this functionality to all inheriting classes. As shared functionalities vary widely from one class to another, this framework rather implements **mixins**, that can be seen as multiple inheritance. Mixins are small classes

that gather a reduced set of very specialized functionalities. Classes required to implement these functions inherit from these corresponding mixins.

These paradigms thus enable a clear comprehension of the code, a class-by-class debug and validation and an enhanced maintainability. Hence, having implemented a specific function only once the framework makes updates of the code easier.

3.2.2 A three-leveled structure

The architecture of the framework, is thus built as a nested assembly of modules, each with its own functional perimeter. As illustrated by figure 3.2, one can organize them along three different hierarchical levels. First, low-level structures and utilities implement functions such as basic data management and computation, some I/O[3] procedures and common neural network functions.

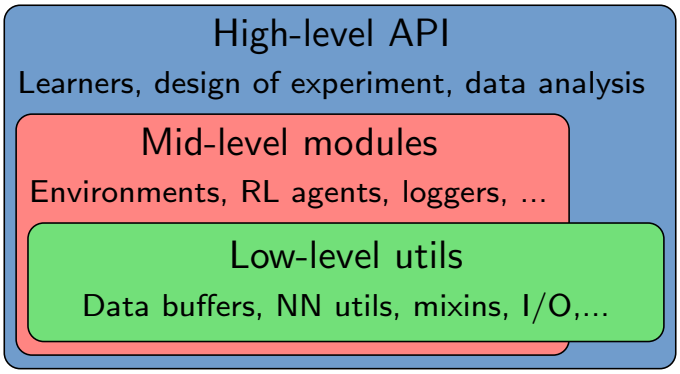


Figure 3.2: RLFramework's three level object hierarchy design

Mid-level modules embody environments, learning agents or model reduction objects. Each of these types of module is defined by its own interface. High-level API sits on-top of this structure and provides a simplified set of functions to the user. These learners, to which external data science modules are added, are the conductors of the mid-level objects. They manage training, testing and data-persistence thanks to predefined scheduling or triggers computed at runtime.

3.2.2.1 Low-level API: data management structures and other utils

Among low-level utilities, data buffers are crucial to the reliability and performance of the framework. They are used by all mid-level modules as well as high-level API. One first finds, scalar (`InfoField`) and vectorized buffers (`LinearBuffer` and `RingBuffer`) that store homogeneous data. These can be write-protected or incrementable and provide miscellaneous updating and accessing functions with the idea of both an efficient storage in memory and a safe and easy access to data. These objects are extended implementations of lists or numpy arrays, that provide handy functions, for instance precomputing miscellaneous statistics or enabling both FIFO[4] and LIFO[5] simultaneously. This latter feature may be needed for action buffers that should provide an ordered action sequence (queue) at the end of a run and enable interpolations from the previous action to the current one (stack).

[3]Input/Output communications, namely reading and writing from memory and asynchronous communication with Message Passing Interface (MPI).

[4]First-In-First-Out, i.e. a queue

[5]Last-In-First-Out, i.e. a stack

These buffers generally need to be assembled together in order to store inhomogeneous data and to be synchronized. This is done by the `Container` class and all its inherited classes later described. This abstraction, similar to a standard Python `dict`, also enables to easily filter out data, for instance to query one-out-of-n samples, or to get different on-the-fly-computed representations of the data, such as lists, arrays or dictionaries. More specifically, `GAEContainer`s extend this class, by providing automated computation of the return (R_t) and the advantage ($A^\pi(s_t, a_t)$) at the end of each episode and facilitated storage of data sequences coming from different environments, that should not be mixed-up. At last, `ReplayBuffer`s provide mechanisms of importance sampling in the buffer, that are used by some *off-policy* learning methods. These are implemented using a segment tree, capable of computing statistics in $\mathcal{O}(n \log n)$ time and that can be queried in $\mathcal{O}(\log n)$ time.

Other structures such as *Space*s that embody the different search spaces (state space or action space) are used by both environments and agents, to check for data compliance and for random sampling. Basic functions enabling to create NN structures, such as multi-layer perceptrons, or Tensorflow nodes computing standard estimators, are also part of these low-level utilities, alongside configuration management routines that solve potential conflicts and ensure standardized as well as backward-compatible naming conventions. One can also name random generator functions or command-line logging verbosity and stack-trace recovery. All these functions can be seen as small building blocks, whose behavior has been once and for all debugged and that are used by higher-level modules.

3.2.2.2 The environment interface

Environments are incarnated by mid-level objects whose class complies with the environment interface. As explained before, this interface is specified by an Abstract Base Class (named `Env`) that defines the minimal set of methods environments should implement (`reset`, `step`, `render`, `seed` and `close`). As functional class inheritance is also used, some of these have a default behavior, that avoids a systematic redefinition in the inheriting class that can be redundant if this behavior is standard. Standard data accessors such as `observations`, `actions`, `reward` and `config`, alongside properties giving access to `space`s fullfil the interface. The general structure of an object of type `Environment` is described by figure 3.3 (left).

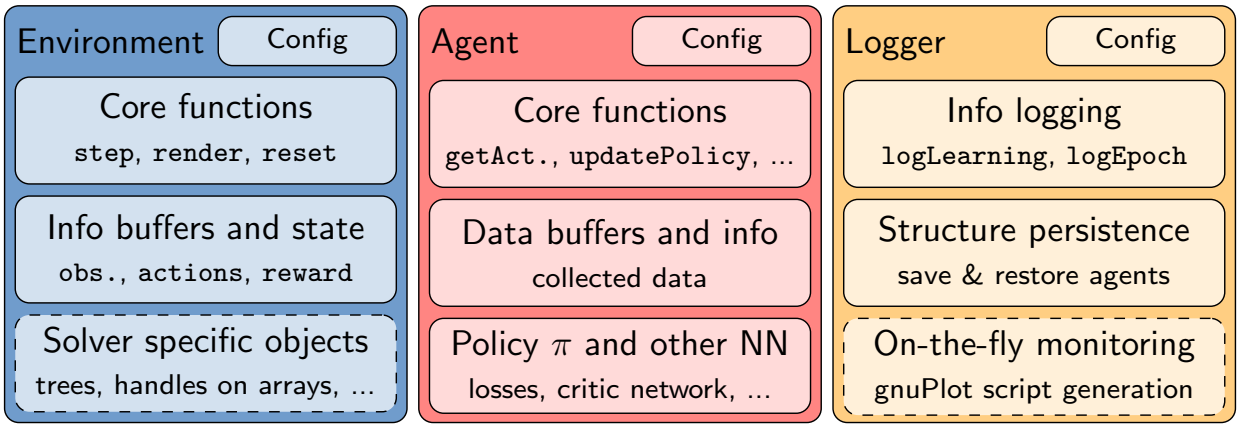


Figure 3.3: Mid-level module interfaces

Specificities of the CFD solvers are handled by mixins and dedicated utilities. Concerning FastS, the most used solver, the `FastsEnvMixin` and `fasts_utils` module gather the methods specific to

handling computation trees, implementing forcing actions and running the solver. This mixin for instance redefines the `step` method and provides standardized instantiation functions that prepare actuation and measurement functions on the simulation. More details can be found in Appendix 9.1.1. Similar structures have been developed for the other solvers used such as FreeFem++ or FeniCs.

Available environments can be divided into two categories. Simple and cheap test cases, used in standard RL studies such as the inverted `Pendulum`, the `CartPole` (a rigid inverted pendulum mounted on a cart), `Atari` games simulator (one of the standard groups of environments for benchmarking RL algorithms), are used to debug newly implemented agents and perform preliminary hyper-parameter searches. To these test cases can be added the `KuramotoSivashinsky` (implementing 1D Kuramoto-Sivashinsky (KS) equation) and the `Landau` oscillator. Canonical CFD cases, such as the `Cylinder`, the `Cavity`, the `NACA` airfoil flow, the `Jet` (a planar supersonic jet on which screech develops), the `MixingLayer` (an incompressible mixing layer displaying a Kelvin-Helmholtz instability), and the `SuperFlatPlate` (implementing a supersonic (Mach 4) flat plate flow on which 2^{nd} Mack mode instabilities develop), are test beds to demonstrate the potential of RL and GP control algorithms.

3.2.2.3 The agent interface

Two different agent interfaces are defined for RL and for GP agents. The GP agents rely on the evaluation of a population of candidate policies. This operation can be parallelized differently than for RL agents, since each individual can be evaluated independently. On the other hand, optimization procedures may require large computational overheads depending on the genetic algorithm flavor. Thus, the mechanics of a GP agent has been built around a task-oriented paradigm, where different kinds of operations (population evolution or individual evaluation) are scattered on the available computing resources. More details can be found in Appendix 9.1.3.

The RL agent interface is likewise defined by an abstract base class (named `RLPolicy`) that specifies the exposed methods (`getActionsTrain`, `getActionsEval`, `updateExpData`, `updatePolicy`, ...) that manage the storage of the collected data, the sampling of control actions and agent optimization, as shown by figure 3.3 (center). Required attributes, mainly concerning information about the state of the agent, are also defined. A batch-splitting method is also implemented, aiming at avoiding large consumption of RAM during gradient-propagation operations that require the computation of arrays of size $\mathcal{O}(n^2)$, n being the batch size. This way, large batches are split into multiple sub-batches that are all sequentially used for training but require less memory overall.

RL agents are categorized into standard, pre-existing and mature ones, such as `DDPG`, `(D)DQN`, `PPO`, `PPO-CMA` and its newly developed sensor-sparse versions, `SAC`, `TRPO`, `TR-PPO-RB` (a flavor of PPO), and "draft" algorithms that are under development, whose stability and reliability is not guaranteed, such as action-sparse versions of `PPO-CMA`, miscellaneous altered versions of the same `PPO-CMA`, with for instance Hindsight Experience Replay [3], spatial attention or LSTM mechanisms, model-based agents such as `PETS-MPC` or modified versions of `APHYNITY`. Implemented GP agents are a standard LGPC learner and its gradient-enriched GMLC [56] version.

3.2.2.4 Other interfaces

Other interfaces have been developed, such as the one for model reduction (named `NN_module`). This interface is mainly dedicated to providing a fixed set of rules concerning the inputs and outputs of Tensorflow objects in order to enable the cascading of multiple modules seamlessly. These modules

implement commonly used model reduction methods using Tensorflow, that can directly be integrated in computation graphs. One can list methods such as matrix decomposition methods (POD, DMD, and Singular Value Decomposition (SVD) methods in general), SINDY [36], clustering (k-Nearest Neighbors), primary and digital capsules used in Capsule Nets [96, 201] or parameterizable LSTM and Transformers.

The logger interface has been defined as a way to unify both RL- and GP-specialized logging functions. This interface provides epoch- and evaluation-logging methods and a generic `saveCheckpoint` method enabling model persistence, as it can be seen in figure 3.3 (right). This interface is derived by both `RLLogger` and `GPLogger` classes that implement these functions according to each specific training method.

3.2.2.5 High level API and wrappers

As introduced, high level API aims at providing the user with a simplified set of functions for creating, loading, training, testing and saving a test case. Concerning RL test cases the `SingleAgentLearner` class handles these tasks. This class enables to create a test case from an initial configuration or from a restored case (whose configuration can be altered on some points). This test case can then be trained simply by calling the `train` method or evaluated (using `evaluate`) or base-lined (using, you guessed it, `baseline`). Stress-tests quantifying the robustness of CFD solvers to different actuation layouts can also be led, to preemptively guarantee that solver crashes are avoided during training.

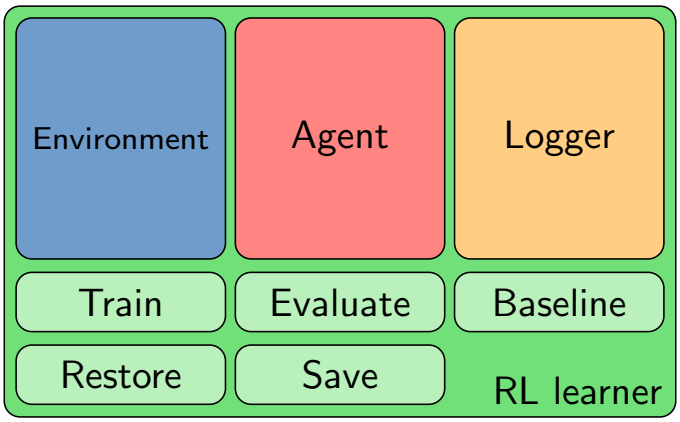


Figure 3.4: Schematics of a RL wrapper, its main methods and its functional mid-level objects.

GP test cases are handled by the `SingleLGPCLearner` that implements the same functions. As detailed earlier, the `train` function can be considered as an heterogeneous task scheduler, running policy evaluation tasks and population evolution tasks alternatively, contrary to what is implemented for RL cases. Again, more details can be found in Appendix 9.1.3.

Alongside high-level learners, a module has been developed to seamlessly generate batches of test cases, exploring the hyper-parameter space. The `DesignOfExperiment` module enables to specify, create a test batch and launch its training in less than a hundred code lines. This class manipulates test case configurations, that can be represented as nested dictionaries or hierarchical trees of hyper-parameters defining all the characteristics of the learner as well as of all the mid-level objects (i.e. number and type of actuators, learning rates, logged information, etc.). Starting from a base test case configuration, automatically complemented by default values, one can explore the effect of one or multiple hyper-parameters by declaring them as dimensions of the problem and by providing the values to explore. This pipeline avoids lengthy, error-prone and useless over-specification of

unchanged parameters. Once specified in a script, a folder containing one sub-folder per test case with the corresponding configuration (stored as a JSON file) and all necessary scripts for launching the training on HPC clusters are generated. Restoring already trained cases instead of starting from scratch is also handled automatically.

Training can be monitored using a `DataAnalysis` object. This class provides methods to manage the folder-based structured created by a `DesignOfExperiment` (or `DOE`) object as well as simplified routines for filtering, grouping test cases according to their configuration. These functionalities to compute ensemble statistics easily. Test cases from multiple test batches can be loaded simultaneously. As loading the full learning data from a potentially large number of test cases can be useless or unpractical, a prefetching system has been implemented to avoid excessive or redundant I/O operations and monitor the amount of data loaded in RAM. Learning curves (possibly involving on-the-fly computations on loaded data) as well as evaluation runs or logged epochs of training can be plotted with a minimal effort of code from the end-user. This class also enables a simplified management of the different checkpoints backups that can be produced along training.

Thus, a standard study pipeline consists in developing a new environment or agent, testing it on a local machine on cheap and well-known environments to get rid of the majority of the bugs and finding a first range for all the hyper-parameters, then running a larger hyper-parameter search (produced by a `DOE` object) on the target environment requiring more resources, possibly on a cluster and monitoring the training using a `DataAnalysis` object.

Policy distillation, introduced in section 2.2.7, has been extended to an all-to-all knowledge exchange paradigm (instead of a one-to-one) and implemented in a specific wrapper. For more details refer to appendix 9.1.4.

3.2.3 Hybrid computing

HPC clusters are organized in nodes that can be seen as individual computers containing a few dozen CPUs (from 28 to 48 on ONERA's clusters) connected with one another. To take advantage of the computing resources available on these clusters, a parallelized training mode has been implemented. This mode enables to run cases in a hybrid MPI/OpenMP paradigm.

3.2.3.1 MPI/OpenMP parallelization

By design, training in parallel mode wraps the test case into a MPI communicator, tasked with transferring data between each stakeholder of the process. This communicator provides an abstraction for communications irrespective of the hardware architecture the process runs on and represents the first level of parallelism. This communicator contains a predefined number of MPI ranks (or processes) that accomplish the tasks they are assigned to. They are synchronized and communicate with other ranks through this MPI communicator.

As shown on figure 3.5, parallelization is handled by a high-level `MultiAgentLearner`[6]. This wrapper handles the MPI communicator thanks to a specific MPI mixin (the `MPILearnerMixin`). The main rank (generally rank 0) hosts the agent, a logger tasked with I/O and a `envMPIcommunicator` (also inheriting the MPI mixin) that "fakes" the presence of an environment on the main rank and that renders sequential or parallel computing indistinguishable from the point of view of the agent. Other ranks host one environment and a logger tasked with collecting information from the environment and sending it to the main logger. Similarly, from the point of view of each of the

[6] or a `MultiLGPCLearner` for GP test cases

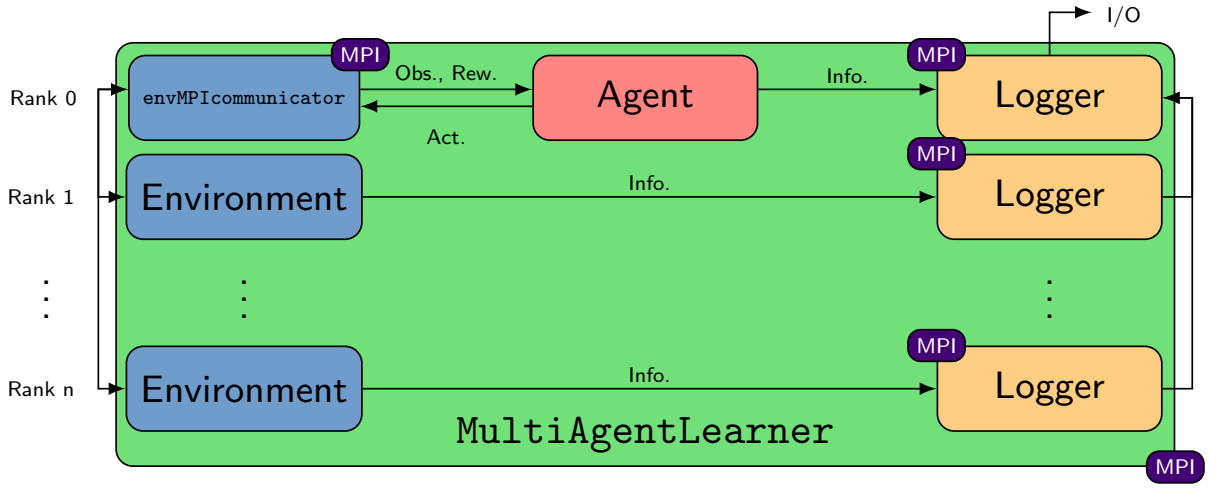


Figure 3.5: Schematics of the `MultiAgentLearner` wrapper. Objects tagged with "MPI" inherit the `MPIlearnerMixin` that provides synchronization routines, MPI communication wrappers as well as properties indicating the roles of the ranks.

environments, the `envMPIcommunicator` acts as an agent. To this object are delegated agent and environment querying tasks during training, normally devolved to the `SingleAgentLearner`. More details can be found in appendix 9.1.2.

Thus here, n parallel environments running independently from each other require $n + 1$ MPI ranks and, as shown by figure 3.6, for a fixed number of control steps or roll-outs, one can choose to parallelize training differently to adapt to the available resources.

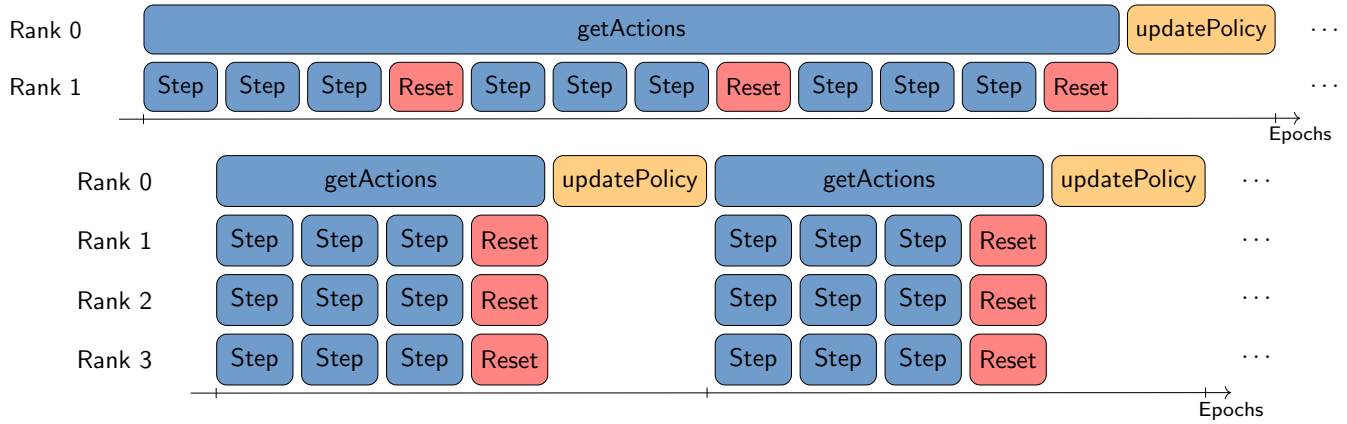


Figure 3.6: Two different resource allocation schedules for a training process using 3 roll-outs per epoch. (top) A 2-rank splitting, requiring reduced computational resources but having a longer training epoch duration. (bottom) A 4-rank splitting, accelerating training at the expense of a larger CPU allocation.

Each MPI process is decomposed into multiple computation threads. This second layer of parallelism is generally transparent in the code and for the end-user. Each environment and agent runs then on a single MPI rank that is multi-threaded using the OpenMP interface previously introduced[7]. This multi-threading is leveraged by the CFD solver as well as the Tensorflow API

[7]As environment computation needs remain low, these can run on a single node and this architecture is valid.

and the numpy library to accelerate computation. As inter-thread communication requires memory-sharing, threads of a same rank must physically run on the same node, but on different CPUs (considering that a one thread-per-cpu pinning is generally enforced). This constrains a rank to be pinned a single node.

If the node is not saturated, other ranks can be affected to it, provided that enough CPUs remain available. Two ranks on the same node will communicate via the MPI standard abstraction but may use faster memory-sharing protocols[8], whereas if located on different nodes, they will resort to other slower protocols and connection buses between nodes. Again, thanks to the MPI standard, this is transparent for the user, except concerning performance.

3.2.3.2 Jobs and hardware architecture

The current section contains voluntarily simplified discussions about parallelism and hardware/software pinning. More details can be found in the documentation of the Intel® MPI library utilized here.

Computation tasks such as running training of batches are submitted to the cluster via a scheduler (here SLURM) in the form of jobs. This is the largest computation unit. The scheduler allocates a given number of CPUs or nodes for a predefined maximal duration. One job can contain the training of multiple test cases, and uses a finite set of CPUs located on predefined nodes of the cluster. The objective of using these resources in the most efficient way possible, relates to an optimization of the matching between the available hardware (computation capacity and network connectivity) and the different software tasks to run, knowing that the MPI ranks' resources cannot be scattered across multiple nodes.

The job inherits a global communicator, spanning all the MPI ranks, across all the test cases. The communicator previously introduced at the scale of a single parallelized test case is in reality spawned as a sub-communicator from this global communicator, as shown by figure 3.7. This brings us to the third level of parallelism, which is more of a convenient way to launch completely independent training processes, rather than a necessity caused by synchronization needs.

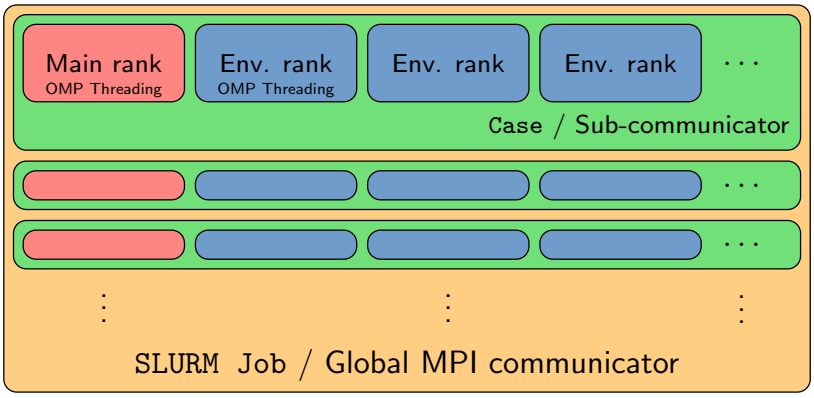


Figure 3.7: Parallel architecture of a multi-case job.

For the sake of simplicity, one generally considers that each CPU processes one single computation thread. This simplifies the thread/rank pinning computations performed by the launching

For large environments requiring more nodes, a MPI splitting of environments will need to be developed, involving a sub-communicator.

[8]depending on the communication fabric

script generated by the `DOE`. This script runs computations that take the threading, number of environments, maximum number of nodes per job and test cases to run as input and outputs command files read by the scheduler to queue one or several jobs immediately or withheld by a job chaining condition (i.e. waiting for the termination of a previously scheduled job to start).

3.3 Other tools developed on purpose

3.3.1 Specific solver boundary conditions

Two specific boundary conditions, mimicking the behavior of jet actuators have been developed and implemented into FastS. These simulate, the injection (blowing) of a prescribed mass flow rate in a specific direction and with a specific total enthalpy and the suction of a prescribed mass flow rate across a tagged domain boundary. This way, blowing/suction actuators can be simulated by alternatively changing the nature of the corresponding boundary condition in the simulation. These conditions have been implemented in Fortran for the low-level computations, using the theory of characteristics and a Newton descent for some conditions. Linking with C++ mid-level functions has also been done in order to expose these new features to the end-user.

3.3.2 Meshing utilities

To complement Cassiopee meshing functions, a small module, implementing basic parametric meshing utilities has been developed. `Meshing Utils` makes possible to mesh 2D or simple 3D computational domains in a parametric way, without resorting to proprietary meshing software. This module can for instance be easily integrated into an optimization loop, running fully in Python. This module enabling fast mesh prototyping and convergence was used to mesh all the geometries studied presently. As well as for the RLFramework presented earlier, a comprehensive documentation has been developed.

3.4 Main test cases

The current section introduces the environments on which RL and LGP algorithms as well as the proposed method are tested.

3.4.1 Non-fluidic test cases

3.4.1.1 1D Kuramoto-Sivashinsky equation

Parts of this section are drawn from Paris *et al.* [175].

The Kuramoto-Sivashinsky (KS) equation is a well-studied fourth-order partial differential equation exhibiting a chaotic behavior and describing the unstable evolution of flame fronts [229]. On a periodic domain of length $L = 22$, the KS equation reads:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial^4 u}{\partial x^4} + a = 0,$$
$$\forall\, t : u(0,t) = u(L,t),$$

where a is the forcing action. For $L = 22$ (the length value used here), the KS equation exhibits three fixed points (named E1, E2 and E3) in addition to the trivial fixed point E0 ($u_{E0}(x) = 0 \ \forall x \in [0, L]$) and low-dimensional instabilities similarly to some low-Reynolds number Navier-Stokes flows, as illustrated by figure 3.8. This test case is interesting for the development and study of flow-control oriented methods because it is computationally inexpensive, allowing to perform large hyper-parameter searches. It also enables to study sensor and actuator locations (with respect to the control target) thanks to its homogeneity along the x-axis.

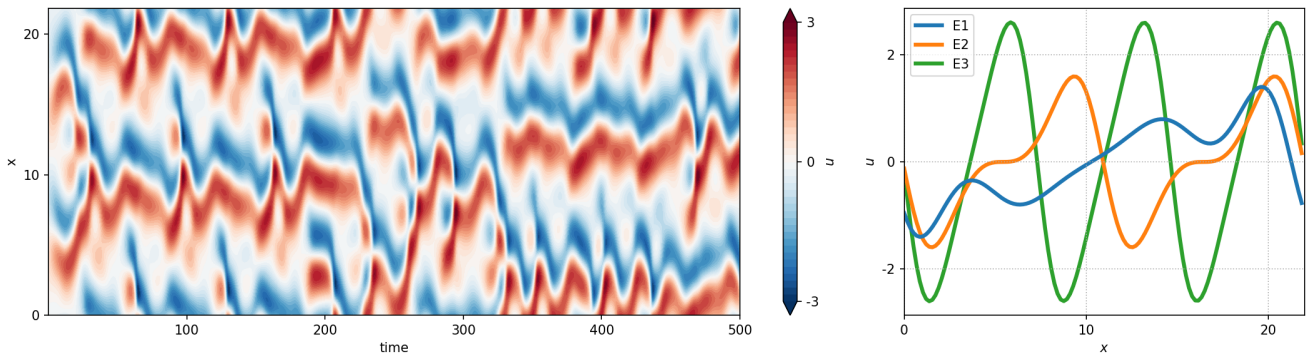


Figure 3.8: (left) Spatio-temporal representation of the dynamics of the KS equation on 500 non-dimensional time units, corresponding to 2000 control steps. (right) Shape of the three fixed points of the KS equation.

The numerical simulation is carried out using a 64-mode Fourier spatial decomposition and a third-order semi-implicit Runge-Kutta scheme (implicit formulation for the linear terms, explicit for the non-linear term) marched in time with a time-step of 0.05. This numerical setup is based on the work of Bucci *et al.* [38] and a code from pyKS. Standard control forcings are designed to mimic spatially localized Gaussian forcing actions:

$$a(x,t) = \sum_{i=0}^{n-1} a_i(t) \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(x - x_i^{act})^2}{2\sigma^2}\right),$$

where n is the number of control actions, x_i^{act} $i \in \{0, ..., n-1\}$ the locations of the centers of Gaussian kernels and a_i the amplitude of each forcing implemented around x_i. The standard forcing action has 8 forcing components implemented at locations ($x_i^{act} \in \{0, 1, ..., 7\}L/8$), $a_i \in [-0.5, 0.5]$ and $\sigma = 0.4$. Partial state observations are provided by measurements of u interspersed between control action locations so that $x_i^{obs} \in \{1, 3, 5, 7, 9, 11, 13, 15\}L/16$, as illustrated by figure 3.9.

A control step is made of an update of the forcing action a_t, then 5 time-steps and the measurement of the observations and reward. A standard run of the KS equation lasts for 1000 control steps. The reset state is seeded using a Gaussian noise of amplitude 0.01 and ran for a random number (from 40 to 100) of control steps without control action, so that control starts on a fully developed instability.

For the present study, the aim of the control is to stabilize u around a pre-defined fixed point,

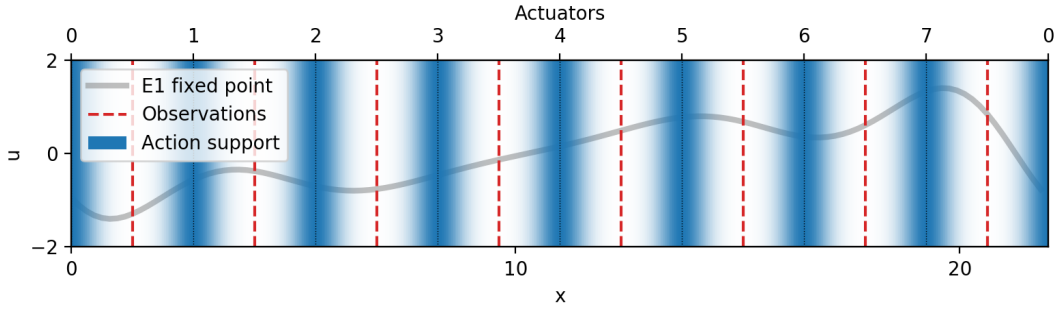


Figure 3.9: Location of observations (measuring u at these locations) and Gaussian supports of the control actions overlaid on fixed point $E1$.

and therefore the reward $r_{t,E}$ is defined as:

$$MSE_{t,E} = ||u(\cdot,t) - u_E||_2 = \sqrt{\frac{1}{L}\int_0^L (u(x,t) - u_E)^2\, dx},$$
$$r_{t,E} = \frac{MSE_{ref} - MSE_t - 0.1||\underline{a}_t||^2}{|MSE_{ref}|}$$

where u_E describes the corresponding fixed point (refer to figure 3.8 (right)), $MSE_{ref,E}$ is the time-averaged reference mean squared error of the uncontrolled state and $\underline{a}_t$ is the control action at time t. This way r_t ranges over $]-\infty, 1]$ and the average baseline (no control) reward is null.

3.4.1.2 Other test-cases

Zero-dimensional test environments have been used to debug and make preliminary validation cases. Among these one can cite the Cart-pole. This environment, taken from the Gym API and modified, simulates a rigid, inverted pendulum mounted on a sliding cart. The pendulum can swing in a single plane and the agent applies lateral forces on the cart in order to hold the pendulum vertical. If x and θ are respectively the position of the cart (with an arbitrary origin) and the angle of the pole with the vertical axis, the dynamics reads:

$$A = \frac{\cos\theta}{m_{pole} + m_{cart}}\left(f + m_{pole} L \frac{\partial\theta}{\partial t}^2 \sin\theta\right)$$
$$\frac{\partial^2\theta}{\partial t^2} = \frac{g\sin\theta - A}{L\left(\frac{4}{3} - \frac{m_{pole}}{m_{pole}+m_{cart}}\cos^2\theta\right)}$$
$$\frac{\partial^2 x}{\partial t^2} = A - \frac{m_{pole}}{m_{pole} + m_{cart}} L \frac{\partial^2\theta}{\partial t^2}\cos\theta,$$

where m_{pole} and m_{cart} are the masses of both the pole and the cart, L is the length of the pole, and g is the gravitational acceleration. Observations are the full-state (position and velocity of the cart as well as angle and angular velocity of the pole). The reward can be varied but is generally either an indicator of the pole remaining within a pre-defined angular range (discrete reward) or a distance to a target angle (usually 0, continuous reward). This environment mainly aims at checking that the implementation of RL algorithms and proposed variants display the same training performances as they are supposed to.

The real damped Landau equations have been developed as another validation environment, for their oscillatory dynamics, similar to multiple flows that present either mixing layers or Von Kármán vortex streets. These equations involve two variables x and y such as:

$$\begin{aligned} \frac{\partial x}{\partial t} &= \sigma * x - y, \\ \frac{\partial y}{\partial t} &= \sigma * y + x + f, \\ \text{with} \quad \sigma &= 1 - x^2 - y^2, \end{aligned}$$

where f is a forcing action. This equation has the unit circle as limit cycle. Control aims at stabilizing both variables around 0 which is an unstable equilibrium point. The reward is defined as a function of the Euclidean distance of the point (x, y) to the origin $(0, 0)$. Similarly to the Cart-pole, these environments are stepped forward in time using simple discretization schemes and are very cheap to run.

3.4.2 Low-Reynolds Cylinder flow

Parts of this section are drawn from Paris *et al.* [174].
This studied configuration is a two-dimensional flow past a cylinder. The geometry is made non-dimensional by setting the cylinder diameter D to 1. The center of the cylinder is located at the origin $(0, 0)$ of the flow domain. Figure 3.10 displays the computed flow domain, which spans over $10D$, and shows the orientation of axes x and y.

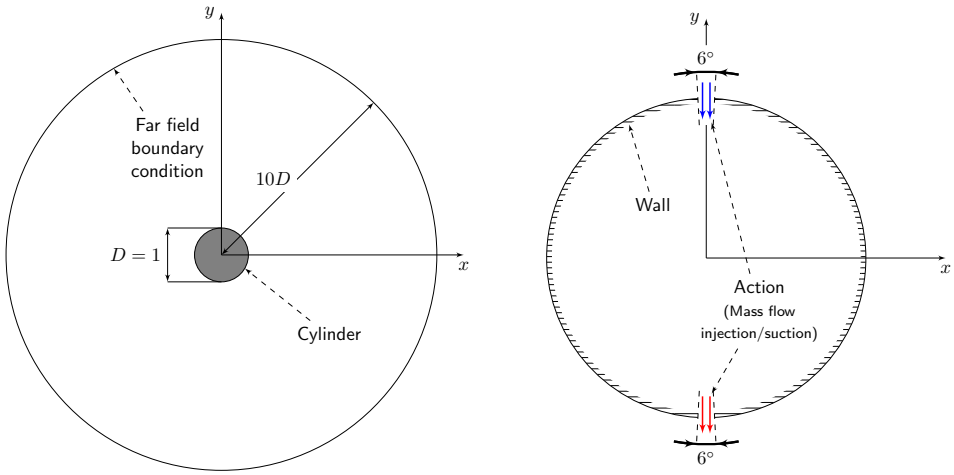


Figure 3.10: Flow domain geometry. (left): Full domain, not at true scale. The far field boundary condition is a characteristic-based inflow/outflow boundary condition modeling free-stream flow. (right): Standard boundary conditions on the cylinder, refer to section 4.2 for more details.

3.4.2.1 Numerical setup

The flow is described by the compressible Navier-Stokes equations. The free-stream flow is uniform at a Mach number M_∞ of 0.10, oriented along x. In the following, all quantities are made non-dimensional by the characteristic length D, the inflow density ρ_∞, the velocity U_∞ and the static temperature T_∞. Note that the Mach number is very low, such that the density fluctuations in the

Parameter	Symbol	Value	Comment/Reference
Flow simulation setup			
Mesh nodes (aziumtally)	-	360	-
Mesh nodes (radially)	-	70	-
Temporal scheme	-	BDF2	Curtiss & Hirschfelder [58]
Numerical time step (non-dimensional)	dt	5×10^{-3}	-
Action ramp length	-	20 it.	-
Maximum action amplitude (non-dimensional)	-	2	-

Table 3.1: Additional numerical parameters

whole domain are negligible, and the flow is therefore quasi-incompressible. The Reynolds number Re, defined as $U_\infty D/\nu$ (ν being the kinematic viscosity), may be varied, but the reference configuration considers $Re = 120$. The flow field is computed via direct numerical solving using FastS for both steady and unsteady computations with a simplified second-order-accurate AUSM+(P) scheme described by Mary [153] for the spatial discretization and a second-order implicit Euler scheme for time stepping. For unsteady computations, a global numerical time step $dt = 5 \times 10^{-3}$ is chosen. Steady solutions are converged using a local time-stepping strategy. Additional numerical details are available in table 3.1. The C-shaped structured mesh is made of 25200 nodes and is refined in the vicinity of the cylinder. The boundary conditions are specified in figure 3.10.

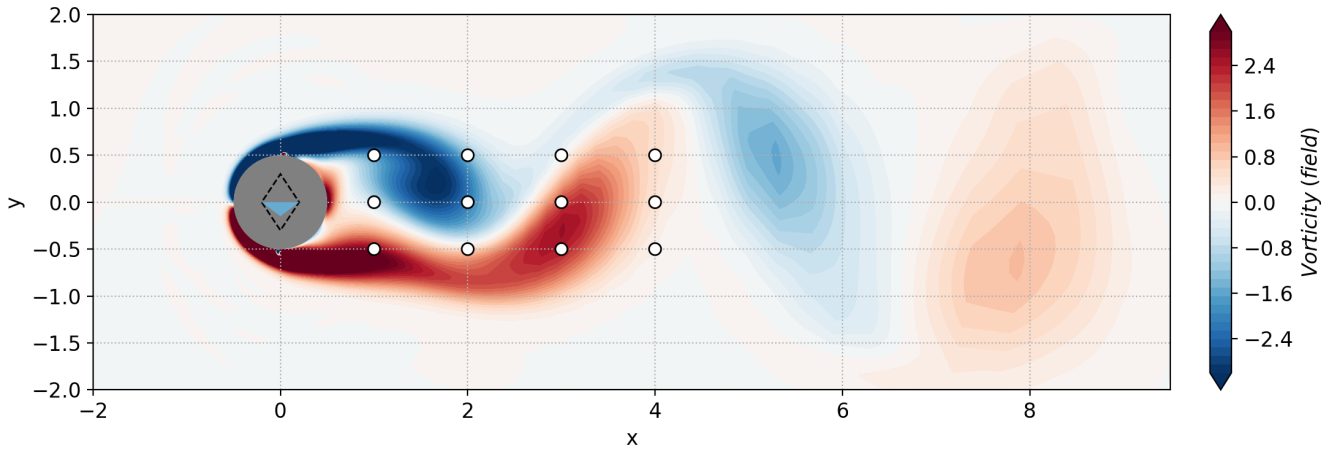


Figure 3.11: Instantaneous vorticity flow field with action $a_t = -0.15$ at $Re = 120$. White dots represent the sensor locations (in a standard layout). The colored triangle in the cylinder depicts the action, its height and color representing its amplitude and sign. The dashed diamond shape marks off maximum actions (both positive and negative). Refer to section 4.2 for more details on sensors and actuators.

Both drag and lift coefficients (C_d and C_l) are computed on the cylinder via the resulting force of the flow $\underline{F}$:

$$\underline{F} = \int_{\text{cylinder}} \underline{\underline{\sigma}}.\underline{n} dS, \tag{3.1}$$

$$C_d = \frac{\underline{F}.\underline{e}_x}{\frac{1}{2}\rho_\infty U_\infty^2 D}, \tag{3.2}$$

$$C_l = \frac{\underline{F}.\underline{e}_y}{\frac{1}{2}\rho_\infty U_\infty^2 D}, \tag{3.3}$$

where $\underline{n}$ is the unitary cylinder surface normal vector, $\underline{\underline{\sigma}}$ is the stress tensor, $\underline{e}_x = (1, 0)$ and $\underline{e}_y = (0, 1)$. These coefficients are used in most of the performance metrics later defined in RL control cases.

3.4.2.2 Uncontrolled flow

The uncontrolled flow configuration (also referred to as the "baseline flow"), displays a well-documented vortex shedding behavior [255] that appears for Reynolds numbers above 46 and which is due to a Hopf bifurcation where the steady solution of the Navier-Stokes equations (the base flow) becomes unstable. Thus, the flow becomes unsteady and follows a stable limit cycle associated with vortex shedding, as shown by figure 3.11.

As presented in figure 3.12, values of the drag coefficient C_d and Strouhal number (defined as $St = fD/U_\infty$, with f being the vortex shedding frequency) have been computed for a wide range of Reynolds numbers to ensure consistency with other studies [169, 31, 255, 93, 90, 21]. For $Re = 120$, the drag coefficient is 1.379 with fluctuations of amplitude 0.018, and $St = 0.18$, which is in agreement with the literature [11, 227]. Note that, since the simulation solves the 2D Navier-Stokes equations, the flow remains laminar across the studied Reynolds number range and does not undergo any additional stability bifurcation.

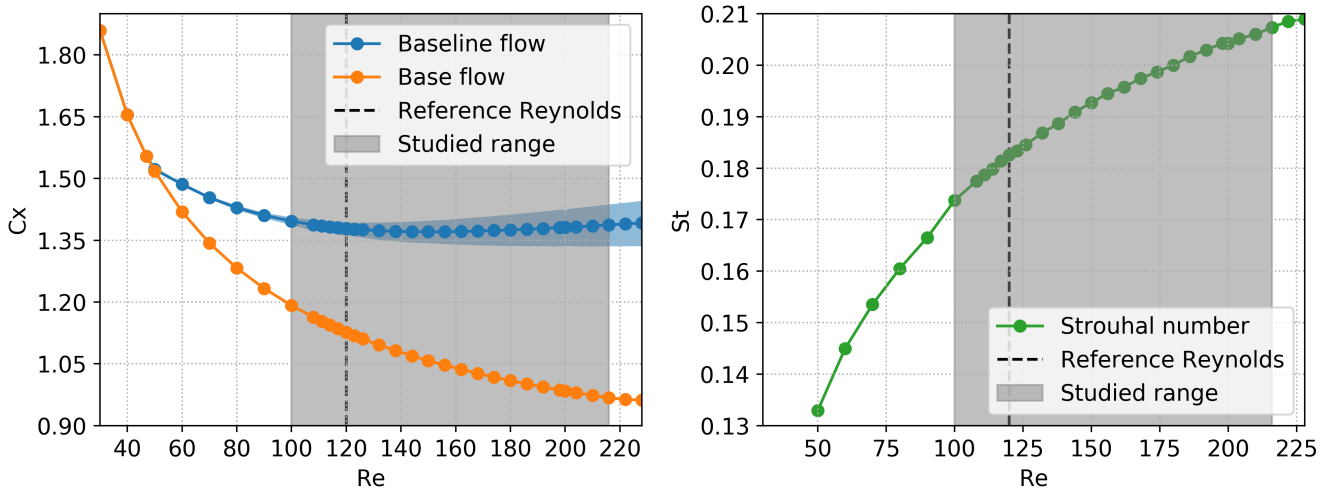


Figure 3.12: (left): Evolution of the time-averaged drag coefficient of the baseline flow (blue) and the base flow (orange) with the Reynolds number. The blue shaded area indicates the variation range of the drag coefficient C_d. (right): Evolution of the Strouhal number of the vortex shedding with the Reynolds number.

3.4.2.3 Potential for drag reduction

Following the work of Protas & Wesfreid [184], the total drag $C_{d,0}$ of the baseline flow can be decomposed into two contributions. The drag of the base flow $C_{d,BF}$, which is constant, and the drag correction due to the flow unsteadiness $C_{d,U}$. If $\langle\cdot\rangle_T$ denotes the time average over a vortex shedding period T, then:

$$\langle C_{d,0}\rangle_T = C_{d,BF} + \langle C_{d,U}\rangle_T \,.$$

Using the base flow performance as a reference, let us define the drag gain μ_{C_d} as a fraction of the drag reduction achieved by the base flow:

$$\mu_{C_d} = \frac{\langle C_{d,0}\rangle_T - C_d}{\langle C_{d,U}\rangle_T}.$$

This quantity measures the relative drag reduction due to the control. Thus, a drag gain μ_{C_d} of 100% is equivalent to a complete suppression of the vortex shedding. Protas & Wesfreid [184] also asked whether a negative mean drag correction $\langle C_d - C_{d,BF}\rangle_T$ could be reached with a periodic forcing, implying a drag gain larger than 100%. Examples from the literature, such as the work of He *et al.* [90] and Tang *et al.* [237], who achieved $\mu_{C_d} = 108\%$ and $\mu_{C_d} = 105\%$ at $Re = 200$, respectively, show that this is possible. However in the case of He *et al.* [90], this performance comes at the cost of a significantly modified mean flow and a large actuation. As shown later in section 4.2, the present study also achieves drag gains slightly higher than 100%, while both preserving the base flow structure and being energy efficient.

3.4.3 Stalled NACA flow

Parts of this section are drawn from Paris *et al.* [175]

3.4.3.1 Setup

This test-case was chosen as a second, slightly higher-regime vortex-shedding flow. A bi-dimensional flow is computed around a stalled NACA 0012 at a chord-wise Reynolds number $Re_c = 1000$. As illustrated by figure 3.13, the origin of the domain $(0, 0)$ is located at the airfoil leading edge, the computational domain is "C-shaped" and extends up to a distance of 20 chord lengths ($C = 1$). The simulation is built in the reference frame of the airfoil, meaning that the angle of attack (α) is imposed by the upstream flow conditions, modeled here as a far-field boundary condition.

Similarly to the cylinder flow, the free-stream flow is uniform at $M_\infty = 0.1$. All quantities are made non-dimensional by the characteristic length C, the inflow density ρ_∞, the velocity U_∞ and the static temperature T_∞. As for the cylinder, the flow solution is computed via direct numerical solving using FastS solver with the same AUSM+(P) spatial discretization scheme, and a BDF-2 time integration. The non-dimensional time-step is set to $dt = 1.3 \times 10^{-3}$ and the structured mesh is made of $120,000$ nodes distributed such that the vicinity of the airfoil and its wake are properly resolved. The boundary conditions are specified in figure 3.13.

The standard control step Δt lasts for 58 numerical time steps of the simulation, corresponding to $\Delta t = 7.9 \times 10^{-2}$ time units. This duration is chosen in agreement with the criteria later discussed in section 4.2.4. Control action is performed on the airfoil suction side through a series of n_{act} independent jet inlets. Negative control actions correspond to suction and positive to blowing, the latter being performed (unless explicitly otherwise mentioned) at an angle of $-80°$ with respect to

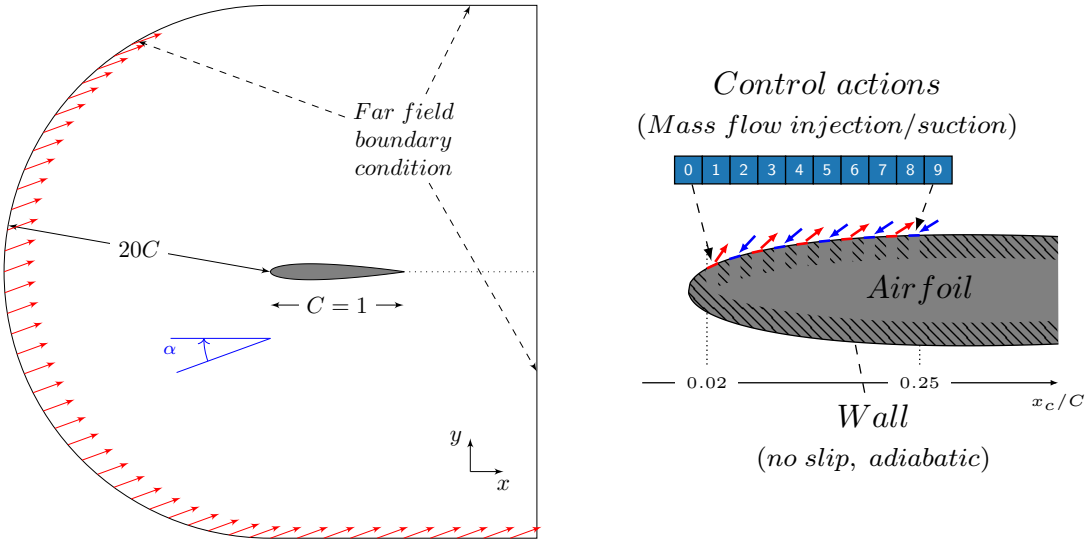


Figure 3.13: (left) Flow domain geometry, not at true scale. α denotes the angle of attack and C is the (unitary) chord length. (right) Boundary conditions on the airfoil, with $n_{act} = 10$. Blue numbered boxes symbolize actuators, number 0 being the upstream-most actuator, number 9 being the downstream-most one.

the local wall normal (refer to fig. 3.13). The control action (ranging $[-1, 1]^{n_{act}}$) is first re-scaled before each control step to the actuators' action limits (here ± 2) and converted to a mass-flow setpoint for the current control step. This command is used to compute the mass-flow boundary condition imposed for each time-step using a 52-iteration interpolation ramp (covering 90% of the control step) in order to avoid abrupt changes that may not be handled by the numerical solver, in a similar fashion as Rabault *et al.* [185] did. Thus, for the i^{th} iteration of control step t, the mass-flow per unit area q_j^i implemented for actuator j reads:

$$q_j^i = \rho_\infty U_\infty \left(a_{j,t-1}^i (1 - r_i) + a_{j,t}^i r_i \right), \text{ with } r_i = \min(i/52, 1).$$

The stagnation enthalpy is held constant (at the upstream flow value) for injection actions. Figure 3.14 illustrates a standard setup for this case.

Again, similarly to what is being done on the cylinder test-case, both drag and lift coefficients (C_d and C_l) are computed on the airfoil via the resulting force of the flow $\underline{F}$:

$$\underline{F} = \begin{pmatrix} F_x \\ F_y \end{pmatrix} = \oint_{\text{airfoil}} \underline{\underline{\sigma}}.\underline{n} dS, \tag{3.4}$$

$$C_d = \frac{1}{\frac{1}{2}\rho_\infty U_\infty^2 C} \begin{pmatrix} \cos\alpha & \sin\alpha \end{pmatrix} \begin{pmatrix} F_x \\ F_y \end{pmatrix}, \tag{3.5}$$

$$C_l = \frac{1}{\frac{1}{2}\rho_\infty U_\infty^2 C} \begin{pmatrix} -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} F_x \\ F_y \end{pmatrix}, \tag{3.6}$$

where $\underline{n}$ is the unitary airfoil surface normal vector, $\underline{\underline{\sigma}}$ is the stress tensor, $\underline{e}_x = (1, 0)$, $\underline{e}_y = (0, 1)$ and α is the angle of attack of the airfoil. Note that lift and drag coefficients are computed by integration around the airfoil on a closed circulation, in the presence of actuators.

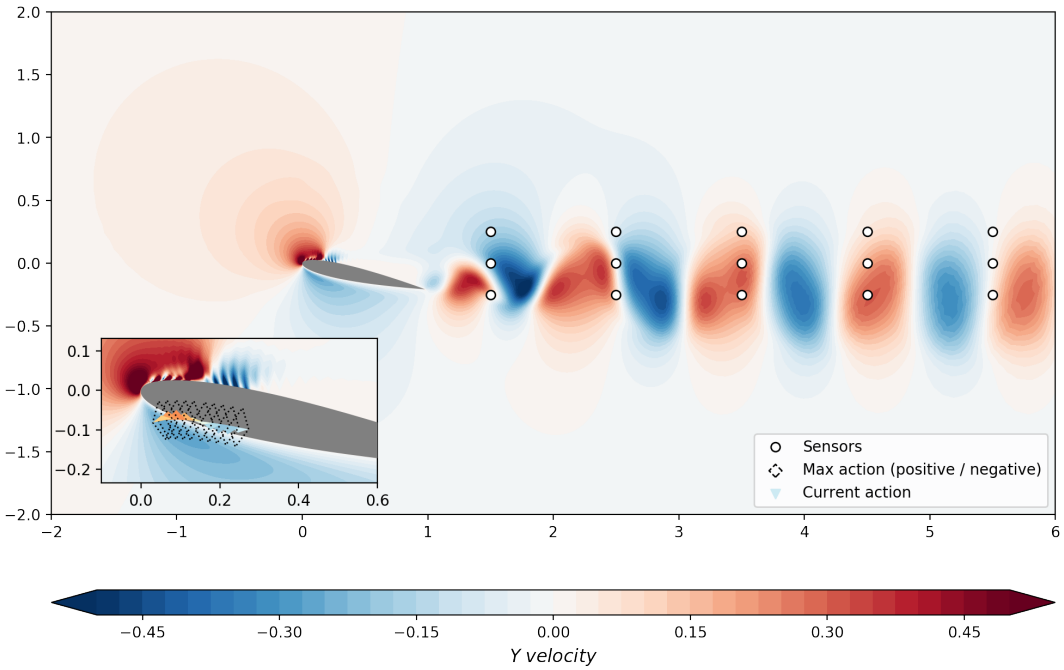


Figure 3.14: Instantaneous Y velocity flow field, with an arbitrary control action (here with $n_{act} = 20$), in the **free-stream reference frame**. White dots represent the sensor locations. The colored triangles nearby the airfoil depict the action, their heights and colors representing each action amplitude. The dashed diamond shapes mark off maximum actions (both positive and negative). The strong variations in velocity in the vicinity of the actuators are due to the presence of interspersed wall boundary conditions in-between actuators.

3.4.3.2 Non-controlled flow

Pre-converged flows are computed for a large number of angles of attack α but the mainly studied flows have α ranging in $[12; 20]$ degrees. At $Re_C = 1000$, the flow is unsteady and displays a laminar vortex shedding [257]. This instability causes both lift and drag coefficients to vary periodically, yielding undesired alternated loads on the airfoil. The evolution of both the amplitude and the periodicity of these loads is shown in figure 3.15. The left-hand side graph shows evolution of the lift and drag coefficients with respect to the angle of attack α. The Strouhal number decreases up to $\alpha = 22°$ and remains nearly steady for larger α values, in agreement with Roshko's correlation formula for a low-Reynolds cylinder [195]. The latter is computed considering the cross-sectional "area" of the airfoil facing the air flow as diameter of an equivalent cylinder. The formula is used in its range of validity since the diameter-based Reynolds number $Re_\emptyset$ ranges from 233 ($\alpha = 12°$) to 451 ($\alpha = 26°$).

Figure 3.15 (right) depicts both lift and drag coefficient spectra for angles of attack of 20° and 25°. For $\alpha = 20°$, both lift and drag coefficients have a very peaked harmonic spectrum with a fundamental Strouhal number $St(\alpha = 20) \approx 0.52$. For $\alpha = 25°$, interspersed peaks appear in-between pre-existing harmonics. The main Strouhal number slightly decreases to $St(\alpha = 25) \approx 0.46$ and a halved Strouhal number St' can be measured at around 0.23. The emergence of the latter for $\alpha \geq 22°$, whose evolution is reported on the left-hand side graph (green dashed line) causes to break the symmetry between two successive shedded vortex pairs.

Figure 3.16 illustrates this symmetry breaking over two vortex periods. During the first period,

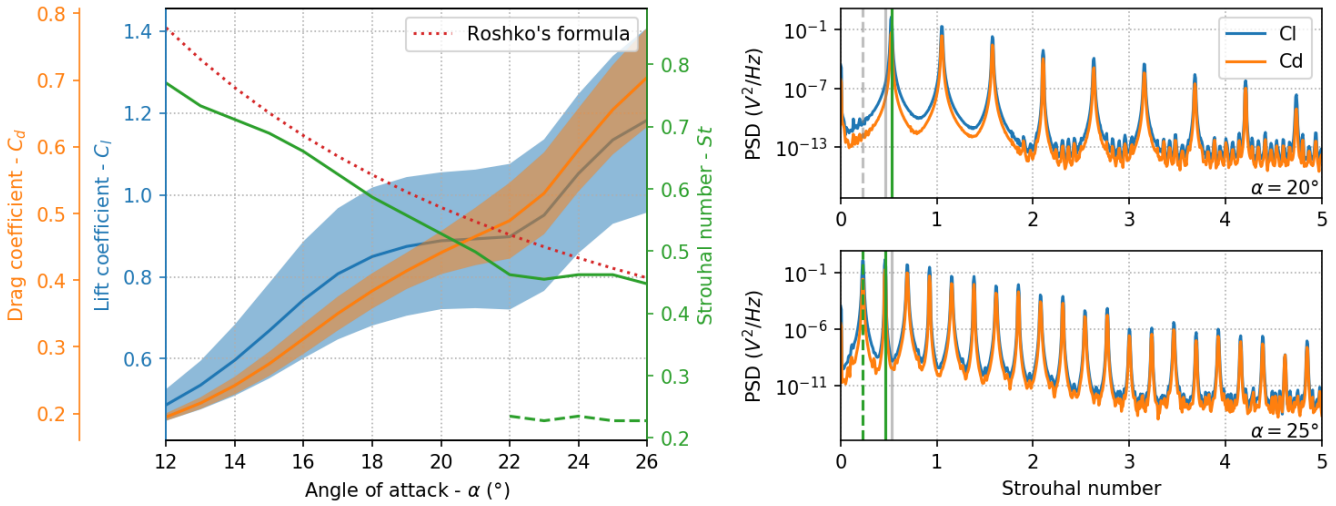


Figure 3.15: (left) Evolution of the lift and drag coefficients and of the Strouhal number with the angle of attack α. The shaded areas depict the standard deviation of C_l and C_d. The secondary Strouhal number emerging for $\alpha \geq 22°$ is drawn in a dashed line. The red dotted line shows the predicted Strouhal number for an equivalent cylinder using Roshko's formula. (right) C_l and C_d power spectral density (PSD) spectra plotted for $\alpha = 20°$ (top) and $\alpha = 25°$ (bottom). Green vertical lines denote the main Strouhal numbers. Other Strouhal numbers (for the other values of α) are reported as gray lines for comparison.

the negative-vorticity, suction-side vortex (colored in blue) induces the shedding of a weak pressure-side vortex downward and feeds a second positive-vorticity vortex on the suction side, near the trailing edge. At the end of the first period, the latter induces an upward shedding of this clockwise-rotating vortex. During the second period, the trailing-edge vortex is stronger and higher compared to its counterpart one period before. It sheds horizontally and stretches the leading-edge vortex induced. Compared to the case $\alpha = 20°$, where trailing-edge and leading-edge vortices have comparable sizes and strengths, at $\alpha = 25°$, the leading-edge then trailing-edge vortices are alternatively dominant over time.

For any angle of attack α, one can define the characteristic period $T = 1/St(\alpha)$ of the unstable phenomenon. This time unit is later used in the study to size the control step. The main goal of the controller is to minimize lift fluctuations using as little control power as possible. Thus, the standard reward (r_t) definition reads:

$$r_t = -S\left(C_l\right)_{2T} - S\left(C_d\right)_{2T} - 0.05\frac{1}{n_{act}}\sum_{i=0}^{n_{act}-1}|\langle a_i\rangle_{2T}|,$$

where $S\left(C_l\right)_{2T}$ and $S\left(C_d\right)_{2T}$ are the standard deviation of the lift and drag coefficients computed over two characteristic periods and $|\langle a_i\rangle_{2T}|$ the absolute value of the averaged i^{th} action component also over 2 periods. Weighting coefficients may obviously be varied for specific experiments.

Lastly one can consider the location of flow separation and reattachment points (with respect to α) as key reference points of the flow topology. These are reported on figure 3.17, alongside the standard, previously described actuator layout. The separation point ranges between 17.3% and 18.2% (of the chord length) for $\alpha = 12°$, between 8.6% and 9.7% for $\alpha = 15°$ and between 3.6% and 4.4% for $\alpha = 20°$. This expected reduction of the chord length of the separation comes along with

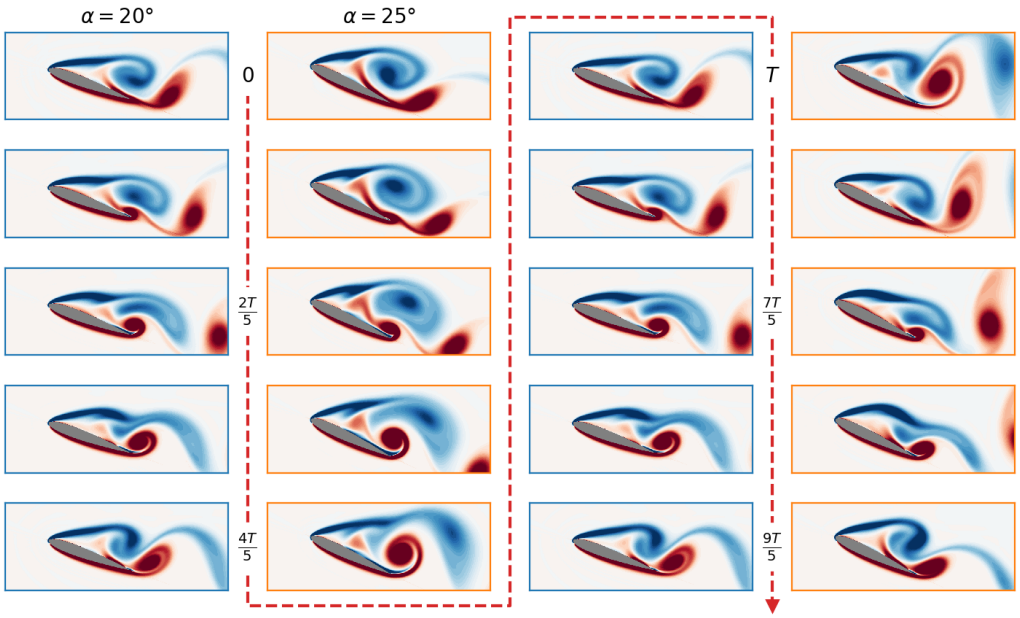


Figure 3.16: Snapshots of the flow vorticity around the airfoil over two characteristic periods T. Blue-framed thumbnails correspond to $\alpha = 20°$, orange-framed ones correspond to $\alpha = 25°$.

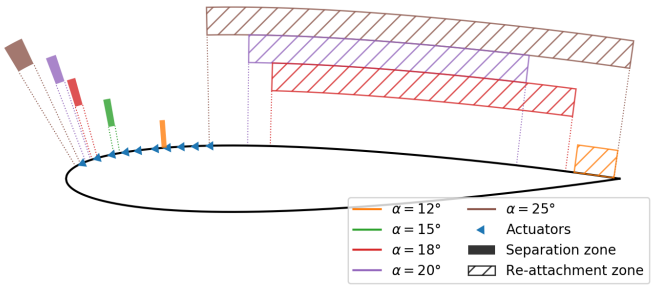


Figure 3.17: Separation (filled areas) and re-attachment (hatched areas) points ranges with respect to the angle of attack. For $\alpha = 15°$, the re-attachment takes place at the trailing edge.

a strong variation of the re-attachment point due to a growing vortex-shedding unsteadiness with α as shown by the figure. For $\alpha = 15°$, no re-attachment is measured, but one can consider it is located at the trailing edge.

3.4.4 Open cavity flow

This section is drawn from an article to be submitted to JFM.
Among the most studied flow control cases, unstable flows past open cavities, occurring in a large range of aeronautical applications and vehicles, are of a particular importance. These fluctuations may develop on any fuselage openings (landing gear boxes or weapons bays), over a wide range of flow regimes and may be critical for airworthiness and safety thanks to alternate loads causing structural fatigue issues, acoustic noise or near wall flow disturbances for instance impacting payload

separation from the main carrier or reduce stealth as emphasized by Cattafesta *et al.* [44]. There is thus a vested interest for efficient and robust control methods for such flows.

3.4.4.1 Phenomenology and existing literature

These unstable cavity flows generally feature a self-sustained Kelvin-Helmholtz-like shear-layer oscillation and a possibly unsteady recirculation region that may act as a delayed feedback to the phenomenon as well as upstream-propagating pressure fluctuations originating from the interaction of the shear layer with the trailing edge.

Passive control on transonic and supersonic open-cavity flows was investigated by Heller & Bliss [92] who assessed the experimental efficiency of passive trailing-edge devices after having described and modeled the dynamics of the oscillations. Later on, Saddington *et al.* [202], Illy *et al.* [103] as well as Yamouni *et al.* [260] and Mettot *et al.* [154] tackled the issue also using passive control. Both identified the importance of the area around the leading edge of the cavity as a go-to location for the tested devices. The latter investigated the effects of a spanwise rod located in this region, disrupting the development of the shear layer instabilities. Conversely at low-speed regimes and in laminar conditions, Liu & Gómez [144] identified the crucial role of the trailing edge region in sustaining the instabilities and that a modification of the geometry of this edge successfully suppresses these oscillations at low Reynolds regimes and on shallow cavities (length to depth ratio larger than 1).

Sarohia & Massier [205] first investigated the impact of continuous injection of fluid at the base of axisymmetric cavities as a mean to reduce radiated noise and shear layer fluctuations. Williams *et al.* [252] and Arnoult *et al.* [6] experimentally investigated the effects of a sinusoidal open-loop forcing using jet actuators blowing from the leading edge on low subsonic flows. They recorded a noticeable reduction of the energetic content of their pressure fluctuation spectra. Guo *et al.* [86] investigated hypersonic regime cavity flows in rarefied gas conditions and identified an important reduction of the maximum wall heat fluxes thanks to the alteration of the flow topology through passive control.

Closed-loop control generally displays greater efficiency thanks to its error-correction mechanism but may be more complex to compute, especially on such cases as pointed out by Rowley & Williams [198] in conclusion of their thorough review on the control of open-flow cavity flows. The computation of the control law generally relies on a linear model of the system dynamics, that can be obtained using different decomposition or projection methods [228]. These linear models were investigated by Rowley *et al.* [199] as well as Sipp & Lebedev [226] who led a thorough global stability analysis of a low regime (around the onset of flow unsteadiness) cavity flow and compared its bifurcation characteristics with the well-studied cylinder flow. Illingworth *et al.* [102] later proposed to model each identified physical phenomenon linearly. Cabell *et al.* [39] proposed high-order models to describe the dynamics and use these to derive feedback controllers actuating a synthetic jet. Adaptive control was experimentally studied by Williams & Morrow [251] to suppress acoustic noise on subsonic flows. Cattafesta *et al.* [43] and later Cattafesta *et al.* [44] used leading-edge piezoelectric flaps as a way to control sub- and transonic flow in a closed-loop fashion. Poussot-Vassal & Sipp [181] introduced a three-step linear model reduction method capable of handling a variation of the Reynolds number of the flow and successfully applied it on a cavity flow.

Barbagallo *et al.* [10], Sipp *et al.* [227] leveraged a Galerkin projection of the Navier-Stokes equations onto a (balanced) POD basis to build a reduced order model of the dynamics of the cavity flow and then performed a H_∞-robust synthesis of their feedback control. They later questioned these choices of model reduction basis and control synthesis criterion in favor of norms and decompositions better suited to the dynamics' spectral content [9]. Dergham *et al.* [63] leveraged a slightly

different model reduction basis using harmonically forced flow snapshots. Standard H_2 synthesis methods were compared to a method combining an ARMAX-based modeling of the flow with a disturbance rejection synthesis by Schmid & Sipp [207]. They showed, that in the case of a cavity, the first one was unable to control the flow satisfactorily while the latter brought an interesting perturbation reduction. Leclercq *et al.* [135] proposed an iterative method to control the flow fluctuations. At each design step of their method, they modeled the controlled flow dynamics using a mean-flow-based resolvent analysis and leveraged structured synthesis method to derive controllers that reduces the global level.

Maceda *et al.* [149] recently proposed a model-free, gradient-augmented genetic approach for discovering experimental control laws capable of damping the energy of fluctuations of a cavity flow by a factor of around 100. They employed a plasma actuator located slightly upstream of the cavity, driven by an amplitude-modulated signal, the amplitude signal being provided by these tested policies. These policies are encoded as a sequence of operations on the measurement signals and are evolved using gradient-boosted genetic selection, mutation and cross-over operations to favor the individuals with the best fitness. Concerning RL and to the best of the authors' knowledge, no study controlling low regime cavity flows have been found in the literature.

3.4.4.2 Setup

This section introduces the control test case and the numerical setup that simulates the flow and the control layout. The studied configuration is a 2-D flow past an open cavity. The cavity is of depth $D = 1$ with a unitary aspect ratio (length is also 1), as shown in figure 3.18. The free-stream Reynolds number $Re_D = U_\infty D/\nu$ is 7500, similarly to Leclercq *et al.* [135], U_∞ being the inflow velocity and ν the kinematic viscosity.

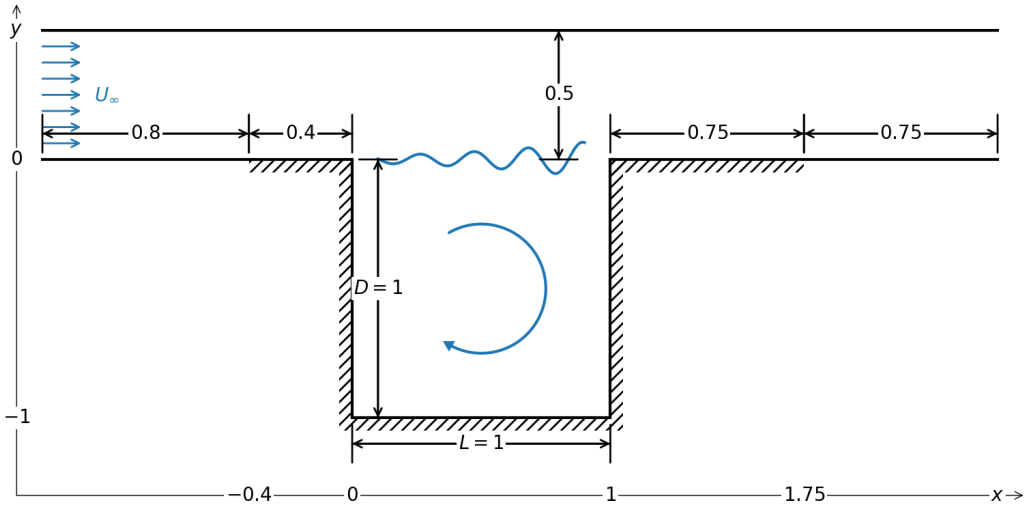


Figure 3.18: Geometry of the cavity. Hatching near solid lines indicate a no-slip wall condition, whereas simple solid lines designate inviscid wall conditions. To inflow is located to the left and the outflow to the right.

A 2-D Cartesian coordinate system whose origin is located at the leading edge of the cavity is used. The inflow is thus aligned with the "x-direction". The geometry of the cavity and the

Parameter	Value	Comment
Mesh size	87400 cells	2-zone, structured mesh
Spatial scheme	ASUM+(P)	refer to Edwards & Liou [69]
Time step	BDF2	implicit second order Euler scheme
Free-stream Reynolds number	7500	
Free-stream Mach number M_∞	0.1	
Cavity aspect ratio	1	$L = D = 1$
Control steps per period	≈ 20	at $Re = 7500$

Table 3.2: Main hyper-parameters

boundary conditions are identical to the setup used by Leclercq *et al.* [135]: the uniform inflow of height 0.5 and velocity $(U_\infty, 0)$ is located at $x = -1.2$ and the boundary layer starts developing from $x = -0.4$ onward thanks to the change in boundary condition later introduced. The trailing edge of the cavity is then located at $(1, 0)$ and the trailing wall stops at $x = 1.75$.

The upstream Mach number M_∞ is 0.1, thus the flow is weakly compressible. All quantities are made non-dimensional by the characteristic length D, the inflow density ρ_∞, velocity U_∞ and static temperature T_∞. The flow is described by the 2-D Navier-Stokes equations and simulated via direct numerical simulation (no turbulence modeling) using FastS finite-volume solver, in a similar fashion as for the previously introduced CFD test-cases. The spatial discretization is based on the modified version of the second-order-accurate AUSM+(P) scheme natively implemented in the solver, and the time integration of the equations is performed with a second-order implicit Euler scheme (BDF2) with a maximum of 10 sub-iterations. The global numerical time step is $dt = 2.74 \times 10^{-5}$ non-dimensional time units. Additional numerical details are available in table 3.2

The structured "T-shaped" mesh is made of around 87400 cells (of which around 10800 cover the cavity) and is refined in the vicinity of both the developing boundary layers and the shear layer over the cavity in both x and y directions to properly capture the dynamics. Inviscid (adiabatic) wall boundary conditions imposing a null normal velocity, are implemented on the top wall and both the extreme upstream and extreme downstream wall portions (i.e. for $y = 0$ and $x \in [-1.2, -0.4] \cup [1, 1.75]$). Both inflow and outflow are simulated using a far field boundary condition. On all the other wall portions a viscous (adiabatic) wall condition, imposing a null velocity is imposed.

3.4.4.3 Uncontrolled flow dynamics

Figure 3.19 illustrated the well-documented dynamics of the uncontrolled flow. One can notice a Kelvin-Helmholtz instability developing in the shear layer than spans the cavity. The literature generally describes two feedback mechanisms maintaining the unsteadiness. First, a fast acoustic feedback due to the emission of acoustic perturbations by interaction of the vortices of the shear layer with the downstream edge of the cavity. A recirculation structure, bringing fluid and perturbations back upstream makes-up a slower, second feedback mechanism to the system. Thus there emerges two different time-scales. Spectra of figure 3.19 display a fast-decaying harmonic content, whose fundamental frequency is linked to the characteristic period of the horizontal velocity fluctuations measured in the shear layer. One may define the reference Strouhal number St_0 as:

$$St_0 = \frac{f_0 L}{U_\infty},$$

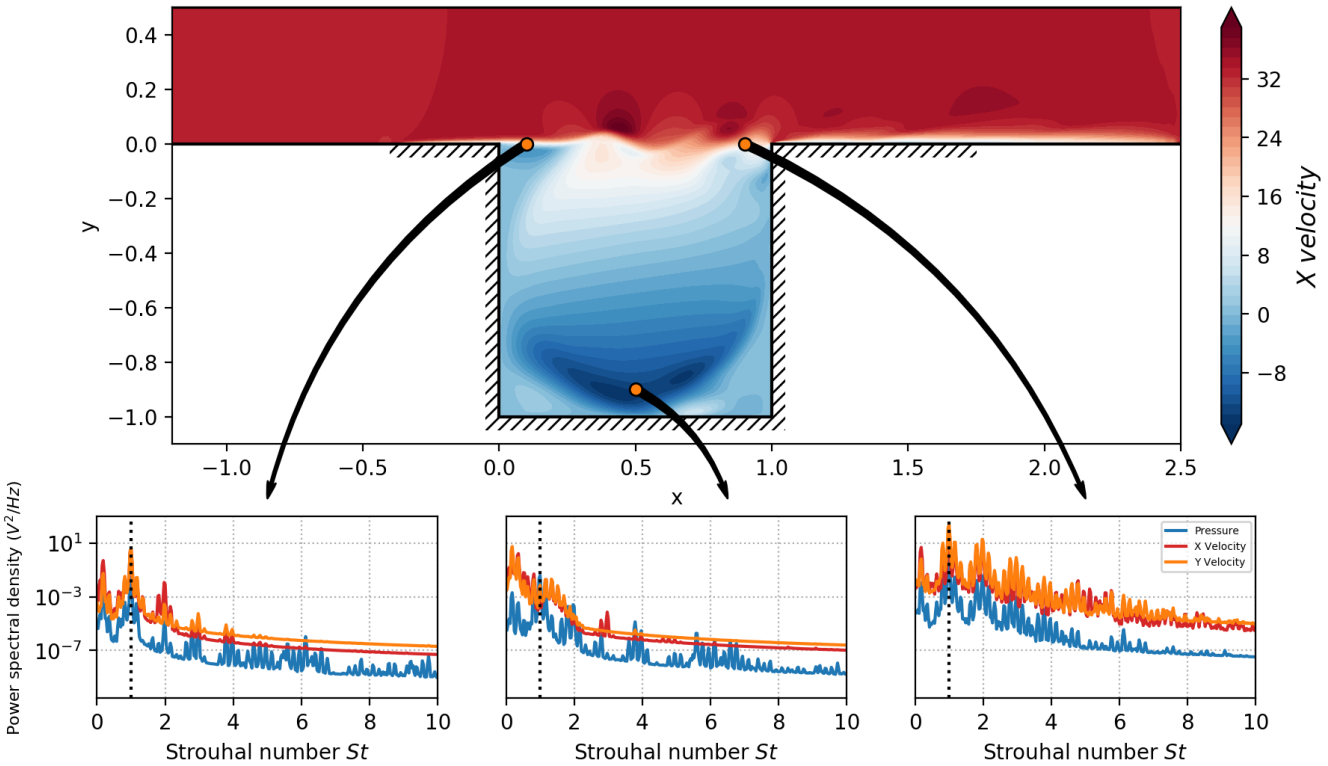


Figure 3.19: (top) Horizontal velocity snapshot of the uncontrolled flow. (bottom) Power spectral densities of the pressure, horizontal and vertical velocities at various locations in the flow. Strouhal numbers are normalized by the reference Strouhal number St_0.

where L and U_∞ are the previously introduced length of the cavity and free-stream flow velocity respectively and f_0 is the frequency of this fundamental harmonic. At $Re = 7500$, $St_0 = 1.82$. Multiple analytic relations providing the dominant Strouhal number(s) from geometric parameters and flow conditions were successively formulated [196, 27, 92, 28] in order to predict the sensitivity of the fluctuation regime to both the geometry and the flow regime.

Chapter 4

RL-based flow control

This first chapter of results presents and discusses the outcomes of the studies led on different test-cases using only marginally modified versions of RL and LGP algorithms. The majority of these modifications concerns hyper-parameter adjustments. These nearly "off-the-shelf" tests help assess the potential of these ML strategy for flow control and highlight fluid-mechanics-specific challenges.

Contents

4.1 Controlling the KS equation

As described previously, the KS equation is a cheap test-case, suitable for quickly prototyping and bench-marking algorithms. In the current section an extensive study of the control on the KS equation towards each of its fixed points is performed. The training of cases toward each of these points is investigated before a study of multi-objective control laws, that are, similarly to the study

of Bucci *et al.* [38], able to handle these four different goals alternatively. The issue of catastrophic forgetting raised by this case of multi-objective RL is also discussed.

4.1.1 Driving the state to its fixed points

Using PPO-CMA and for every fixed point $E0$, $E1$, $E2$ and $E3$, training cases where run with different actuator layouts, as shown by figure 4.1. The actor and the critic are embodied by densely connected neural networks having two layers of 512 neurons each. The chosen activation function is a ReLU, except for the output layer which linearly activated. Unless otherwise stated, this NN architecture is used for all the neural networks in the following (for actor, critics, reduced-order models as well as other estimators), the effects of the size and shape of NNs being kept out of the scope of this study. Here policies are trained on 8 roll-outs of 250 steps each and for 2000 epochs (4 million state transitions in total) for each test-case. Averages are computed on batches of 5 test-cases. The different layouts have 2, 4 or 8 actuators and may be shifted by fractions of L along the x-axis in order to study the effect of their location on the control performance. One can first notice that, as expected, all fixed points are efficiently reached using full (8-actuator) layouts. Here, as the chosen figure of merit is the average reward on a training run, best performances do not reach 1 exactly since the control transient lowers the average reward and the exploration noise (driven by σ, that stabilizes around 5×10^{-2}) that produces slightly sub-optimal trajectories.

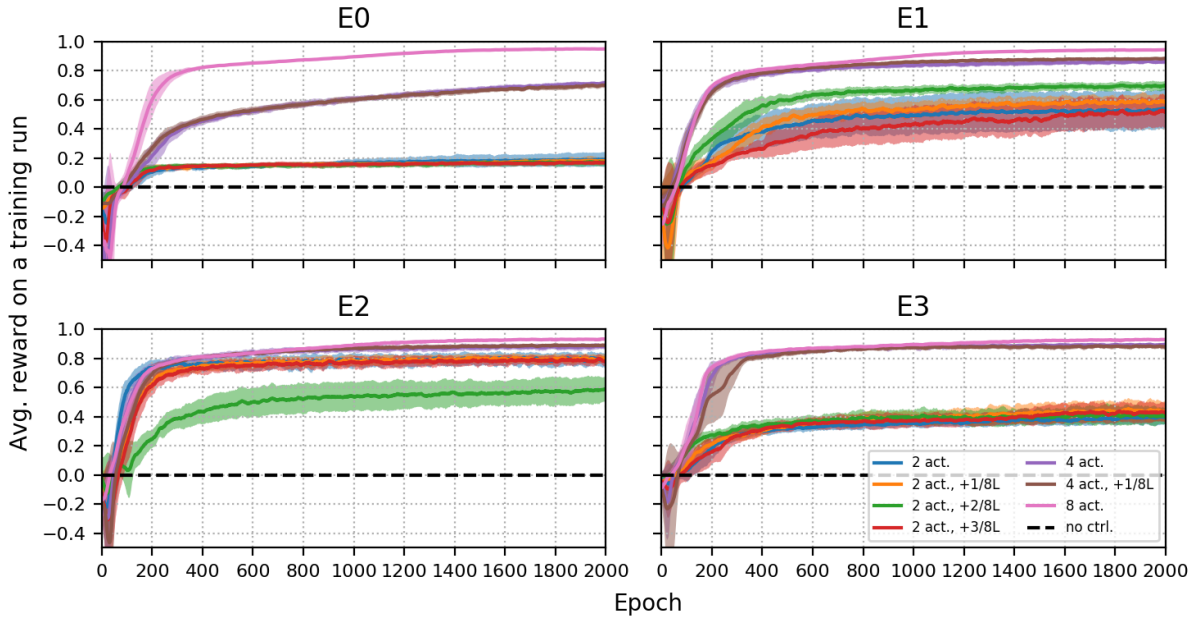


Figure 4.1: Learning curves for different target fixed points and different actuator layouts. Ensemble averages are computed on batches of 5 test-cases. Shaded areas represent standard deviation computed across batches.

Second, it can be seen that 4-actuator layouts perform nearly identically on targets $E1$, $E2$ and $E3$, contrary to target $E0$. Similarly performances of 2-actuator layouts are notable for target $E2$, to the exception of one layout, but decreasingly efficient for other targets $E1$, $E3$ and $E0$. No clear explanation of these observations has yet been proposed, but the observed reproducibility of the results, confirms the impact of actuators position with respect to the topology of the control target.

Concerning fixed point $E0$, as the target solution is invariant along the x-axis, no actuator layout stands out from the others for the same number of actions. For targets $E1$ and $E3$, performance discrepancies in-between 2-actuator layouts can be noticed. From figure 4.1, one can qualitatively rank fixed points from the easiest to the hardest to reach as: $E2$, $E1$, $E3$ and $E0$. These observations, alongside ancillary performance indicators are reported in table 4.1.

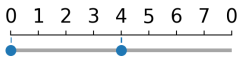
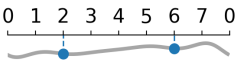
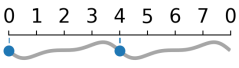
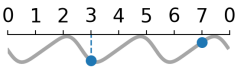
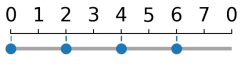
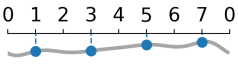
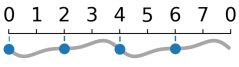
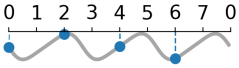
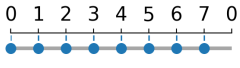
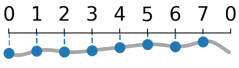
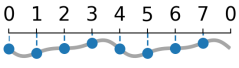
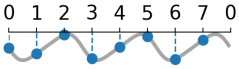
		E_0	E_1	E_2	E_3
2 act.	best layout				
	avg. perf.	28.4%	71.0%	81.0%	53.1%
	time to 80%	–	57	28	65
4 act.	best layout				
	avg. perf.	71.9%	88.7%	89.4%	89.9%
	time to 95%	–	45	41	39
8 act.	best layout				
	avg. perf.	95.1%	94.5%	93.9%	93.7%
	time to 95%	18	17	17	28

Table 4.1: Control performances and best actuator layout with respect to the control goal fixed point and the number of actuators.

As expected, the more actuators, the shorter the control transient (here measured as the averaged number of control steps to reach a given performance) and the better the average reward. Figure 4.2 compares the control performances of randomly picked control laws for each fixed point target with 8-actuator layouts. Both the temporal evolution of the reward and the spatio-temporal representation of the state depicts these short transients, confirming the proficiency of the control.

4.1.2 Multiple goals and catastrophic forgetting

Trainings where the control target is randomly changed with various probabilities have been performed on 8-actuator layouts with the aims of learning a multi-objective policy. Contrary to the previous fixed-objective test-cases, the current target must be provided to the agent for it to adapt the control to the goal. To do so, the observation vector, so far containing only measurements of u at locations interspersed in-between actuators, has been augmented with a "one-hot" encoding of the current target. Then, at every control step (except for test runs), this control goal can be re-sampled with a given probability, that gives the average goal-update frequency. Update probabilities ranging from 10^{-6} (update every 4000 runs on average) to 10^{-2} (2.5 times per run on average) have been investigated.

As illustrated by figure 4.3 average control performances for each of the four fixed points are strongly impacted by the update frequency. Conforming to intuition, cases with the lowest update probability perform very well when the current training target coincides with the test goal, but much more poorly otherwise. This is evidence for catastrophic forgetting. The agents being trained on a single control goal for an extensive number of consecutive epochs tend to "over-fit" their behavior on the most recent training data. As this training data is biased toward one single control

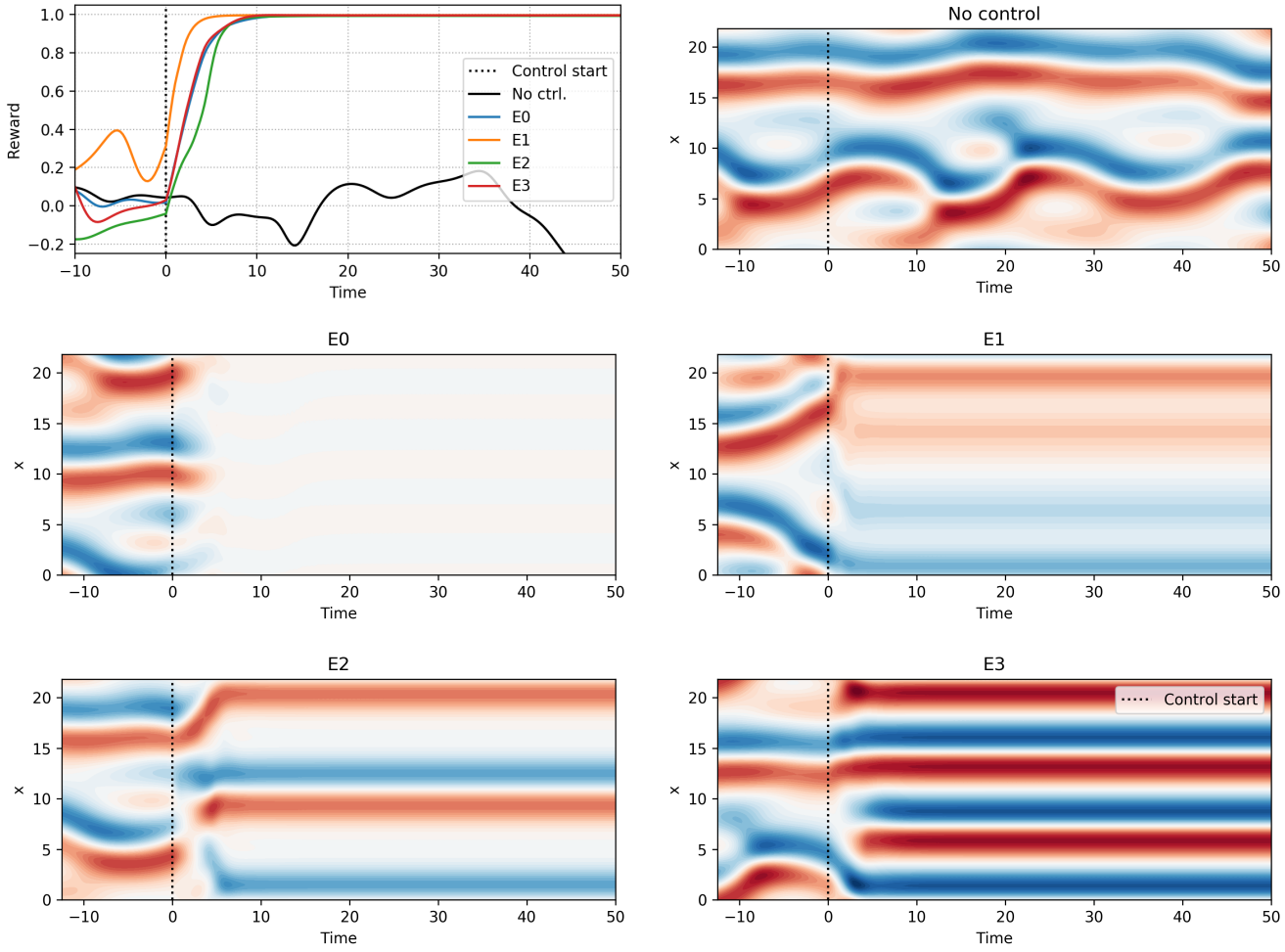


Figure 4.2: Comparison of control laws with different goals and with a non-controlled (baseline) run and with 8-actuator layouts. (top, left) Evolution of the reward depending on the control goal. (top, right) Uncontrolled evolution of the KS equation. (others) Controlled evolutions toward each of the fixed points.

objective, no periodic training signal constrains the agent to keep the same level of performance on other goals. Upon training goal re-sampling (no shown here) the reward collapses to very low levels, before recovering in a similar fashion as agents trained from scratch.

The other extreme sampling probability (every 0.4 run on average) displays much more homogeneous performances that do not depend on the current training goal. As shown in the previous part, 100 control steps are more than enough to converge toward the control goal with 8 actuators. Yet average rewards are lower than the more moderate frequencies of goal re-sampling, especially for target fixed point $E3$. One explanation for it can simply be that despite short transients, there is a data imbalance between the transient and the converged state, where most of the gains can be performed by shortening the transient, possibly at the expense of slightly less optimal converged states. These lower performances could also be explained by the trickier learning of the value function. Here, as the change in control goal is random, the value function is learned on trajectories, containing yet consecutive states, but potentially multiple control goals. Thus here V^π is tasked to approximate $\mathbb{E}_{s\sim T, a\sim\pi}\left[r_{t,E} + \gamma\mathbb{E}_{E'}\left[r_{t+1,E'}\right] + \gamma^2\mathbb{E}_{E''}\left[r_{t+2,E''}\right] + \dots\right]$, where E is the current

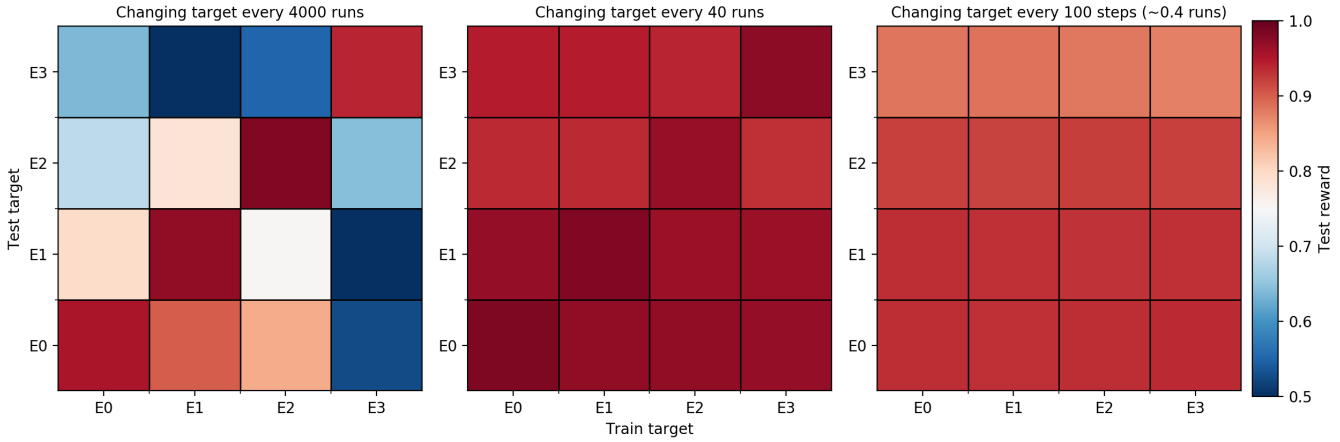


Figure 4.3: Test reward (average on the last 20 steps of an evaluation run), with respect to the test target (ordinate) and the current train target (abscissa) for different frequencies of train target sampling. The target is kept constant for the whole test run. Ensemble averages is performed on 500 test runs for each sampling frequency.

target, and E', E'', etc. are the future ones. Future work could consider the impact of learning goal-conditioned value functions.

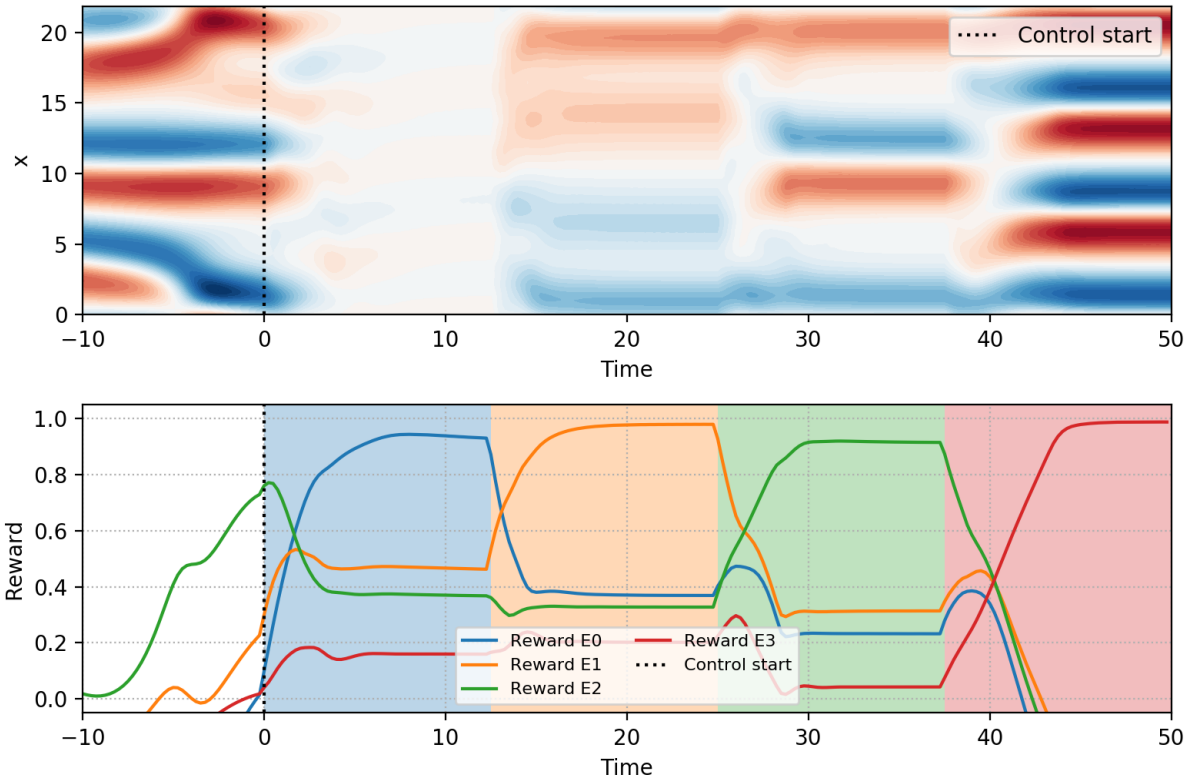


Figure 4.4: Control performance of multi-objective policy trained with an average update period of 40 runs. (top) Evolution of the state throughout time. (bottom) Corresponding evolution of the different rewards with respect to time. Shaded areas in the background depict the current test target.

Finally the remaining cases with an average update period of 40 roll-outs seem close to the optimal trade-off between the two previously discussed extremes (even though slightly better performances can be noticed when both training and testing goals coincide). In this case, the update period is low enough to prevent catastrophic forgetting but high enough to enable the exploration of the best control strategies for each goal. A randomly picked policy having this update frequency has been tested and the four targets successively. Results a reported on figure 4.4 confirm the proficient behavior of the control law.

4.1.3 Control using model-based RL

Model-based RL has been tested on the KS environment using the same 8-actuator-8-observation setup. These results are compared to model-free RL (PPO-CMA) presented before. The PETS-MPC [53] method, as described in part 2.2.6, has been run on 8 roll-outs of 250 control steps per epoch. The first 50 epochs are run using randomly generated control actions, in order to properly pre-train the models. 5 models were trained and control optimization involved batches of 100 actions sequences each evaluated with 15 "particles". The training was stop after 300 epochs.

4.1.3.1 Model-based/model-free comparison

First, and as shown by figure 4.5 (top), the performances display a skyrocket increase from epoch 50 onward (start of the action sampling using the CEM[1] optimizer [29]). The training performance then quickly converges toward its optimal value. Comparing with the training using the PPO-CMA which converges toward the same performances but much more slowly and gradually, the training of PETS-MPC algorithm seems to be done after around 10^5 environment transitions whereas the PPO-CMA requires more than 30 times this amount in data on the full-state environment.

Concerning control stochasticity, PETS-MPC appears to be much more deterministic than PPO-CMA whose σ average value decreases much more slowly. This fact partly accounts for PPO-CMA learning much more slowly than PETS-MPC. Regarding the epoch duration (in CPU time), PPO-CMA costs about 20 times less than the PETS-MPC algorithm which is due to the action inference process of the later requiring extensive model querying. Thus, in terms of CPU time and on a cheap environment both methods perform somehow equivalently.

4.1.3.2 Performances of model-based control

Figure 4.6 illustrates to control performances of the PETS-MPC method. One can first notice that the control goal (fixed point E_1 here) is rapidly reached in around 20 control steps. All the models predict the rise in reward but they are all pessimistic on the longer term evolution whether they predict from step 0 or step 20 onward. Comparing the true observations with the forecast ones at step 0, the models correctly identify the tendency to reach fixed point E_1 but do not consider it as a stable state, since they later foresee strong modifications of the partial state. A step 20, once the fixed point is almost reached, the forecast is much more stable, the destabilization being foreseen in a much further future comparing the prediction at step 0 (around 50+ steps later at step 20 compared to 25 steps in the future at step 0).

This model accuracy is confirmed by figure 4.7. Whether it is for observations or the reward (concatenated as a single prediction vector), model accuracy is very strong on the first few future

[1] Cross-Entropy Method

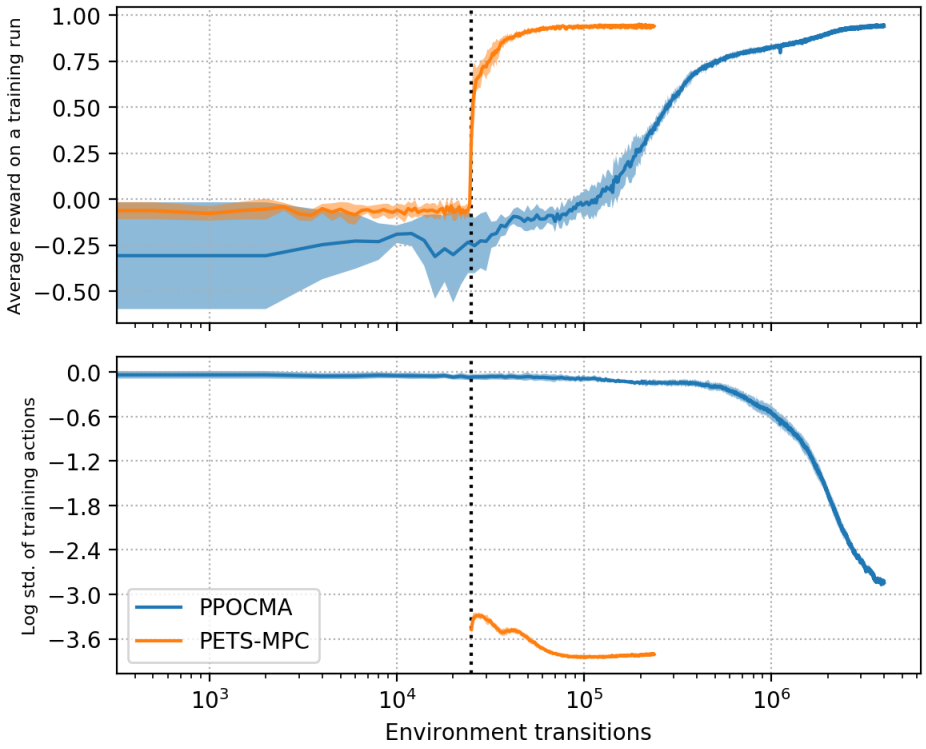


Figure 4.5: Comparisons of the learning curves of the model-based PETS-MPC algorithm with the model-free PPO-CMA method. (top) Average reward during training runs on the full-state environment with respect to the number of observed environment transitions from the start of the training. (bottom) Evolution of the log-standard-deviation ($\log \sigma$) of control actions.

steps then exponentially degrades. Considering the average variance of observations and the reward respectively as reference, one can consider that the predictions of these models are informative up to around 40 time steps in the future. One can also notice that the pessimistic tendency of the models is confirmed here (refer to the left-hand side graph).

This accuracy is to be considered with care, since it is evaluated on data having the same probabilistic distribution as the training one. This means that model dynamics are learned only "around" the control trajectories and that they may not extrapolate properly on data distributed differently. Attempts to increase this prediction ability, notably by training models over multiple prediction steps, using "Neural-ODE-like" training formulations and using auto-encoders to embed the partial-state into a more "dynamics-friendly" state-space have not been successful so far.

4.2 Controlling a low Reynolds cylinder wake

This section introduces the training and performances of RL-trained control on a low Reynolds cylinder flow. The setup of this test case has been detailed in section 3.4.2. **Parts of this section are drawn from Paris *et al.* [174].**

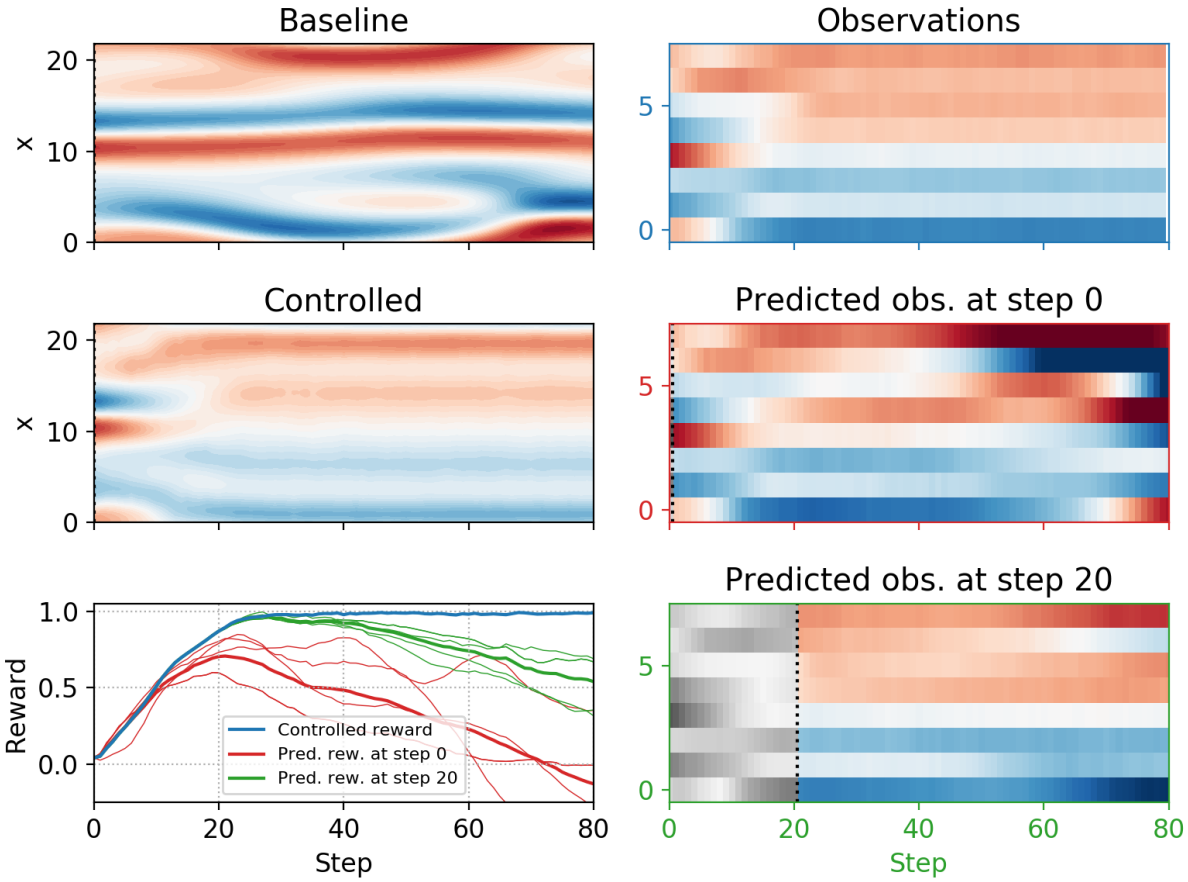


Figure 4.6: Evolution of different metrics during a test run. (top left) Evolution of the non-controlled state. (center left) Evolution of the controlled state. (bottom left) Evolution of the observed reward (blue line) and of the predicted reward at step 0 (red lines) and at step 20 (green lines). Thin lines represent the forecast of each model and thick ones illustrate the ensemble average. (top right) Evolution of the observations. (center and bottom right) Predicted evolution of the partial state at step 0 and step 20 respectively, by an arbitrarily selected model.

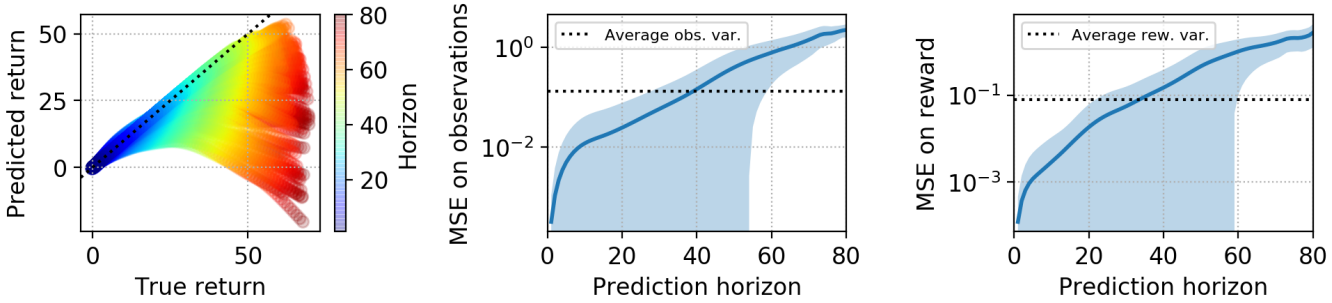


Figure 4.7: Prediction accuracy of the models. (left) Predicted return (undiscounted sum of predicted returns) with respect to the true return. Color indicated the forecast horizon. (center and right) Mean square error on the observation and reward prediction respectively. Average variance of observations and reward is plotted in dashed lines for comparison.

4.2.1 Setup and training

As shown in figures 3.10 and 4.8, actuation is (and unless stated otherwise) performed by injection or suction on the cylinder's poles through two 4° or 6°-wide jet inlets. The control step Δt is defined as the number of numerical iterations during which the control command is held constant. Here, a control step lasts for 50 numerical time steps, thus $\Delta t = 0.25$ non-dimensional time units. This choice and its impacts on training and overall performance are discussed later in section 4.2.4. For each control step, a command a_t (positive or negative) is translated into a blowing/suction using a 20-iteration interpolation in a similar fashion as described for the control of the NACA-0012 flow in section 3.4.3.1. To ensure an instantaneous zero-net-mass-flux for every action, the two poles act reciprocally: $+q_i$ is imposed on the top inlet surface and $-q_i$ on the bottom inlet. Note that in several studies, the actuators are such that they are able to inject streamwise momentum, which may directly reduce the cylinder drag. In the present study, the actuators are designed such that they can only inject cross-stream momentum, thus making any "direct" drag reduction impossible.

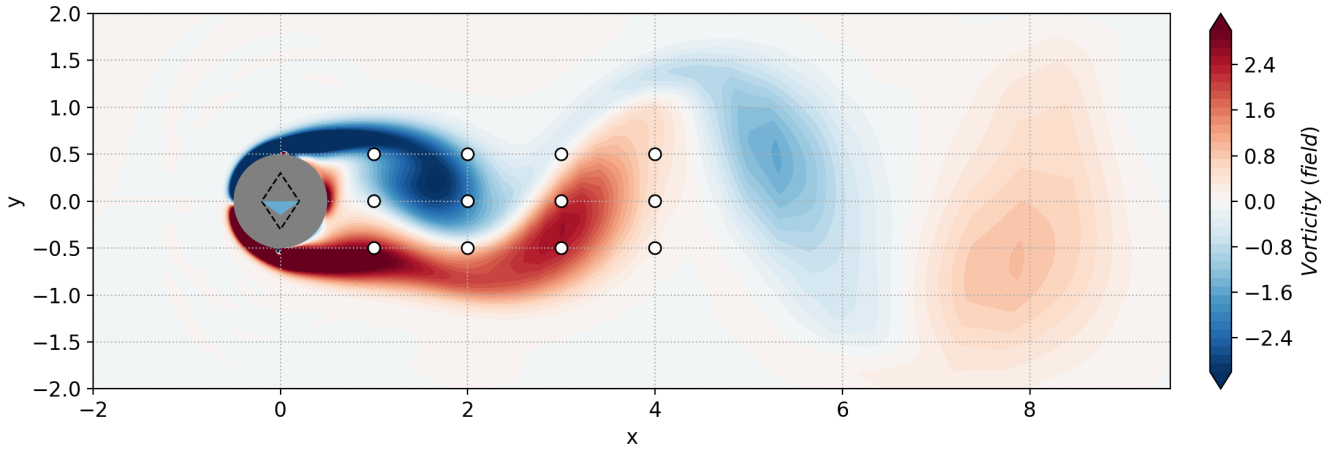


Figure 4.8: Instantaneous vorticity flow field with action $a_t = -0.15$ at $Re = 120$. White dots represent the sensor locations. The colored triangle in the cylinder depicts the action, its height and color representing its amplitude. The dashed diamond shape marks off maximum actions (both positive and negative).

Multiple sensors record the pressure of the flow at predefined locations and at the end of every control step. The standard grid-like sensor layout is made of 12 sensors at locations $(x, y) \in \{1, 2, 3, 4\} \times \{-0.5, 0, 0.5\}$. The output measurement is a pressure fluctuation, defined as the difference between the local non-dimensional static pressure and the reference inflow static pressure p_∞. Figure 4.8 illustrates a standard setup for this case.

4.2.2 Control performance

Unless otherwise stated, all results are obtained using the reference case at $Re = 120$, with the 12-sensor layout described in figure 4.8 and PPO-CMA as learning algorithm. A standard training of 200 epochs of 480 control steps each requires around 180s per epoch on 4 CPU cores. Hence, a training of 200 epochs costs around 40 CPU hours, most of the CPU time being used to run the environment. Figure 4.9 illustrates this learning process. A large variation of mean C_d values can be observed in the first epochs of training, then C_d values concentrate more around their moving average. This is caused by PPO-CMA decreasing the exploratory variance σ when performance

stabilises. For the following results, training is performed sequentially (unlike Rabault & Kuhnle [186] who explored the potential of parallelized learning or other cases discussed in the study).

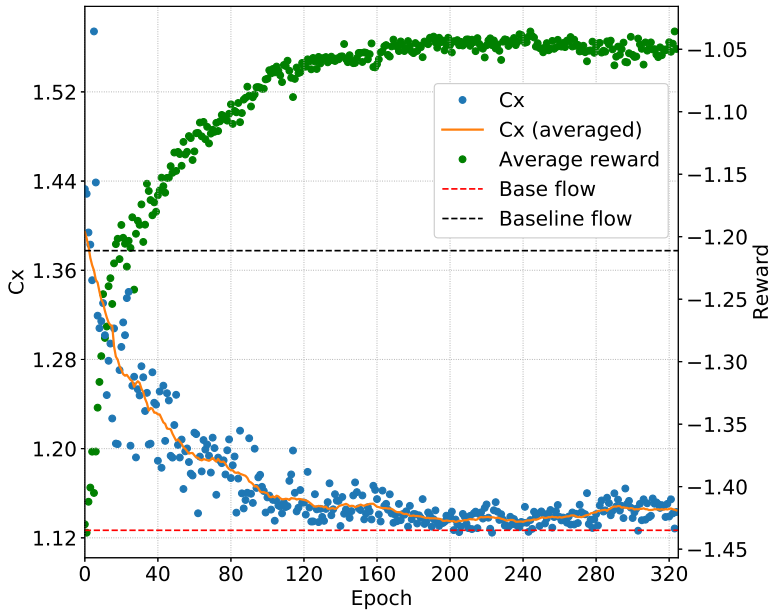


Figure 4.9: Standard learning process. Each C_d value is averaged over the whole epoch, including the transient from developed vortex shedding to controlled flow. This explains the discrepancy with pure performance values on C_d later introduced. The yellow curve is a 20-epoch moving average of C_d values. The average reward (green dots, reward r_t averaged over the current epoch) shows a quasi-monotonic growth that saturates from epoch 200 onward.

4.2.2.1 Control performance and efficiency

At a Reynolds number of 120, the time-averaged baseline flow drag coefficient is $\langle C_{d,0} \rangle = 1.379$. Performance in terms of drag reduction is computed as a percentage of the average baseline flow drag coefficient $\langle C_{d,0} \rangle$ and also using the drag gain μ_{C_d} introduced in section 3.4.2.2. Figure 4.10 shows the instantaneous drag coefficient C_d, the corresponding action a_t and instantaneous lift coefficient C_l throughout control steps. A first phase, from time $t = 25$ (control starting) to $t = 50$ approximately, shows a rapid transient from the fully developed vortex shedding instability to the controlled flow. This transient corresponds to approximately 4.5 vortex shedding periods. During that phase, actions have a large amplitude and do not seem to follow any simple pattern. In a second phase, from time 50 to the end, the drag coefficient is stabilized to a value below $C_{d,BF}$. This represents a drag reduction of about 18.4% and a drag gain μ_{C_d} around 100.6%. Actions have a significantly reduced amplitude compared to the first phase, and they appear to have a slightly non-zero average. Starting from $t = 150$, a periodic action pattern seems to appear in the form of modulated bursts. These last two points are further discussed in the next section.

For other Reynolds number values, drag reduction has also been measured. Results are presented in table 4.2 and confronted to other studies carried out on the same case.

- At $Re = 100$, a drag gain slightly larger than 100% is also reached. The observed mean flow is similar to the base flow.

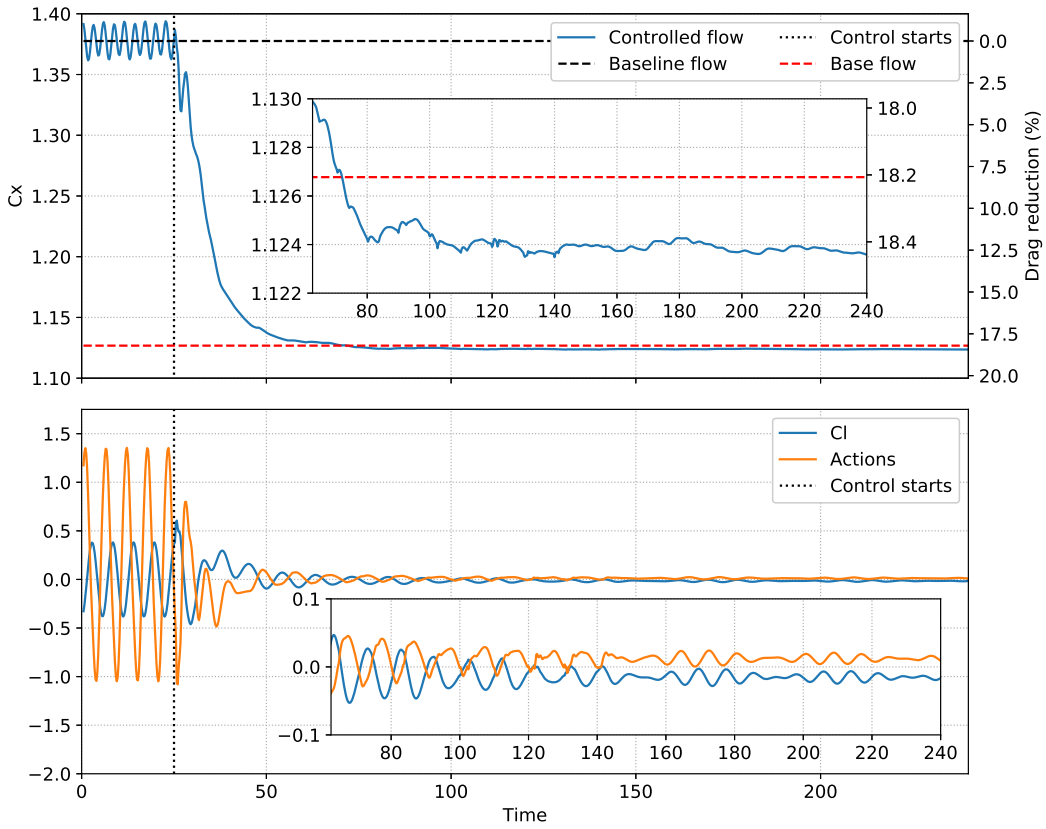


Figure 4.10: Performance of the active flow control strategy. (top): Evolution of the instantaneous drag coefficient C_d. (bottom): Evolution of both action and lift coefficient C_l.

- At $Re = 150$, the net drag reduction is more than 21%, close to the value reached by Protas & Wesfreid [184]. In their case, they used large amplitude rotary open-loop control, causing the controlled flow to remain unsteady. In the case of blowing/suction, a similar open-loop strategy has been tested at $Re = 120$, with far less success in terms of drag reduction. This proves the limitations of open-loop strategies for the current blowing/suction setup.

- At $Re = 200$, the results from He *et al.* [90] slightly outperform those of the present study in terms of drag reduction. But their drag gain is also obtained at the cost of an important mean flow modification, as previously mentioned in section 3.4.2.2.

Some existing studies on the control of the cylinder flow have not been included in table 4.2 due to the significant discrepancies of their test case compared to ours, which prevents any straightforward comparison. For instance, Min & Choi [155], using a 360° blowing/suction actuation, end up artificially reducing the equivalent diameter of their cylinder. While their results and methodology are interesting, comparison of performances is however not relevant here. Atam *et al.* [7] used a 360° suction actuation steered by four different controllers and manage to match a target drag efficiently. The work of Arakeri & Shukla [5], who impose the tangential velocity on the cylinder surface and force a quasi upstream-downstream symmetric flow, is not included in the comparison for similar reasons. Among the related – yet not directly comparable – interesting work, we can also cite Sohankar *et al.* [231] who achieve a significant drag reduction (67%) for a porous square cylinder flow at $Re = 100$, the paper from Muddada & Patnaik [163] which shows rather important

gains (a drag reduction of 53%) using two small rotating rods in the vicinity of the cylinder, or the results from Chen & Aubry [49] using magneto-hydrodynamic forcing to stabilize a cylinder flow at $Re = 200$. For a more exhaustive list on the topic, one may refer to the review from Rashidi *et al.* [188].

Re	Drag red. (%)	Drag gain μ_{C_d} (%)	PSR	Learning type	Action type	Reference
100	8.0	54.6	-	Gradient descent	Blowing	Leclerc *et al.* [134]
	8.0	92.7	-	DRL	Blowing	Rabault *et al.* [185]*
	5.7	67.2	-	DRL	Blowing	Tang *et al.* [237]*
	14	95.5	-	ANN/ARX	Translation	Siegel *et al.* [220] Seidel *et al.* [214]
	14.9	101.7	158	DRL	Blowing	**Present study**
150	4	17.5	-	Tuned open-loop	MHD	Singha & Sinhamahapatra [224]
	15	65.7	51	Adjoint NS	Rotation	Protas & Styczek [183]
	21.3	93.4	0.3	Tuned open-loop	Rotation	Protas & Wesfreid [184]
	21.2	92.9	188	DRL	Blowing	**Present study**
200	**31**	107.9	-	Adjoint NS	Rotation	He *et al.* [90]
	28.6	99.6	0.07	POD-based	Rotation	Bergmann & Cordier [20]
	24.5	85.3	0.26	POD-based	Rotation	Bergmann *et al.* [21]
	21.6	106.9	-	DRL	Blowing	Tang *et al.* [237]*
	28.6	99.6	110	DRL	Blowing	**Present study**

Table 4.2: Drag reduction and performance comparison. *These cases are slightly different since walls parallel to the flow are added. Action types: "Blowing": Blowing/suction on cylinder poles, "Translation": Vertical translation of the cylinder, "Rotation": Rotation of the cylinder, "MHD": magneto-hydrodynamic forcing

Another important indicator of the control performance is the energy required for drag reduction. The instantaneous actuation power is computed as $P_{act} = \sum_{i=1}^{2} |q_i \Delta p_i S/\rho| \approx |a_t| S \rho_\infty U_\infty U_{jet}^2$, considering that compressibility effects are negligible, S being the area of one actuator, U_{jet} the injection velocity (kept constant), Δp the excess pressure yielded by the actuator. No actuator efficiency is considered here. Taking the time-averaged baseline flow drag power ($P_0 = \frac{1}{2}\rho U_\infty^3 D \langle C_{d,0} \rangle$) as reference, the actuation power P_{act} peaks at 3.2% of P_0 in the early stages of the first control phase, but only represents less than 0.1% of P_0 on average in the second phase (see figure 4.11). Thus the total power expenditure – necessary to both counteract drag and implement action –, is temporarily higher than for the baseline flow. But it is quickly counterbalanced by the significant decrease of both drag and actuation powers during the second control phase. In the example shown in figure 4.11, the energy trade-off starts being beneficial 4.25 time steps after the control starts, which is long before the flow stabilization.

The Power Saving Ratio (PSR) introduced by Protas & Wesfreid [184] is defined as the ratio of the gain in drag power to the time-averaged control power $\langle P_{act} \rangle$. In the quasi-steady controlled regime, $PSR \approx 143$ for $Re = 120$, showing that the control obtained here is highly energy-efficient. For other Reynolds numbers, PSR are reported in table 4.2, and the values found are significantly higher than 1 even for the highest Re considered. It is noteworthy that high PSR values are very

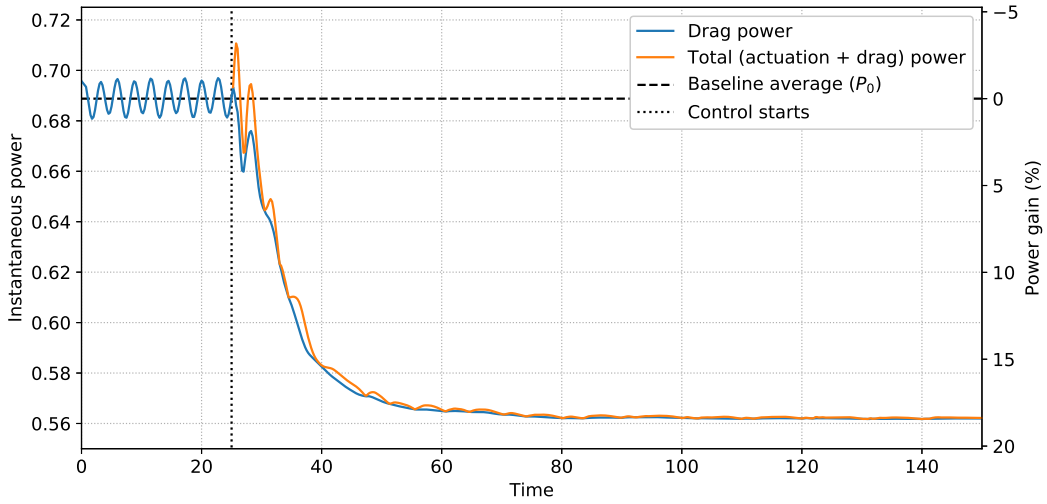


Figure 4.11: Evolution of power expenditure throughout control. The drag power is the power necessary to withstand drag forces, the actuation power represents the power P_{act} spent on action implementation. The power gain measures the instantaneous total power expenditure saved as a fraction of the averaged uncontrolled drag power.

sensitive to $\langle P_{act} \rangle$ as it gets smaller. Thus, only the order of magnitude of the PSR is meaningful in these cases.

It can be noticed that for $Re = 150$, Protas & Styczek [183] achieved extremely energy-efficient control, but with a lesser net drag reduction than the present study. On the other hand, the open-loop controls performed by Protas & Wesfreid [184], Bergmann *et al.* [21] and Bergmann & Cordier [20] reach significant drag reduction but with low PSR values. This highlights the difficulties to get a control law efficient both in terms of net drag reduction and power consumption, which is achieved by the present DRL approach.

Note that quantitative comparisons of PSR values from different studies is often not meaningful. For instance, in the case of cylinder rotation, the actuation power does not consider the inertia of the rotating cylinder (the mass of the cylinder is considered null). In the case of suction/blowing, the actuator efficiency is supposed to be perfect. This highlights that power-based comparisons between different actuation types may have a very limited relevance, and should only be considered quantitatively to distinguish highly efficient control ($PSR \gg 1$) from inefficient ($PSR = O(1)$) strategies.

It is interesting to note that, despite its high energy-efficiency, the control policy is obtained without any explicit penalization of the instantaneous control power. The actuation power expenditure is not directly included into the measured performance during learning. However, the reward r_t is penalized by the time-averaged lift coefficient, which ensures parsimonious actions since a strong action generates strong lift. Even though C_l is averaged over one vortex shedding period, the periodicity of the flow varies (or even vanishes) during training which makes a perfect compensation of the effects on C_l of positive and negative actions very unlikely. As described early on, slightly non-zero-average actions are systematically observed during the second control phase. Thus, a lack of convergence cannot account for this fact. Instead, an increase in the penalization on lift through α causes a reduction of this constant component.

However, an increase in α has downsides. By trying several values within the range $[0; 5]$, it has

been observed that, for $Re = 120$, the chosen value $\alpha = 0.2$ is close to the optimal trade-off between pure performance and energy consumption. For both an increase or a decrease of α, the PSR decreases and the drag reduction shows a very slight decline. The slight negative effect on the PSR when α decreases below 0.2 can be explained by the reduction of the penalization on large actions, thereby increasing the control power expenditure. On the other hand, an overly large value of α reduces the observed exploratory variance due to the strong disadvantage put on large amplitude actions that are necessary in the early stages of the control to achieve a near-stabilization of the flow. The search of the optimal α value has only been performed for $Re = 120$, and the value of 0.2 has been retained for other Reynolds values. Therefore, the PSR values presented in table 4.2 may not be optimal and might be improved by a careful choice of α. But from the results obtained for $Re = 120$, it appears that α is not a sensitive parameter: it leads to negligible changes in drag reduction performance, and for a wide range of α values, the PSR remains significantly higher than 1.

4.2.2.2 Analysis of the controlled flow

A common difficulty with deep learning approaches is the physical understanding of the results. Unfortunately, no simple action pattern has been noticed throughout the evaluations of the control strategy, whether it is for the first or second control phase. Unsuccessful attempts to reproduce this action behavior with simpler linear controllers (simple gain and delayed response based on POD analyses) might indicate that complexity is required to reach the observed control efficiency. The projection of the controlled transient onto the 3 most energetic POD modes of the natural transient only captures at best 57% of the energy of the flow (while they are enough to almost fully describe the natural transient). This shows that the controlled system is significantly more complex than the uncontrolled one, which may explain the encountered difficulties. While it is hard to precisely explain how the control policy acts on the flow to reduce drag, the present section nonetheless attempts to describe the control based on an *a posteriori* analysis of the flow.

As studied by Nair *et al.* [165], who used cylinder rotation or momentum injection parallel to the flow to impose an energy optimal phase-shift control, the drag reduction seen in the transient phase, is caused by the delay in vortex shedding. This generates "elongated vortex structures", that also stabilize the instantaneous recirculation bubble. Similar observations have been made in our case. As shown by figure 4.12, the first phase of the control strategy is a fast transient from fully developed vortex shedding to a stabilized cylinder wake, where the actions trigger the shedding of vortices slightly earlier than the natural shedding. This results into longitudinally stretched and weaker vortical structures.

Once the flow has been stabilized and is nearly steady, its drag coefficient is very close to $C_{d,BF}$. Figure 4.13 compares the convergence of C_d with the length of the instantaneous recirculation bubble. This length is multiplied by more than 2.5 during the control phase and peaks at 99.5% of the base flow recirculation bubble length. The correlation of both the increase of the length of the recirculation bubble and the drag reduction is a well-known fact [184, 185]. The recirculation bubble lengths found in this study are in good agreement with the reference literature [267, 184]. From time step 100 onward, both base flow and controlled flow have a very similar recirculation bubble, as illustrated by figure 4.14. The "tail" of the controlled bubble slowly flaps vertically with a very moderate displacement amplitude ($\Delta y < 0.3$) at $St \approx 0.12$. This confirms that the control policy tends to lead the flow toward the base flow, the latter being an unstable optimum with respect to drag. The controlled flow reaches a small amplitude cycle around this equilibrium point, driven by a slowly modulated, small amplitude, quasi-periodic control action.

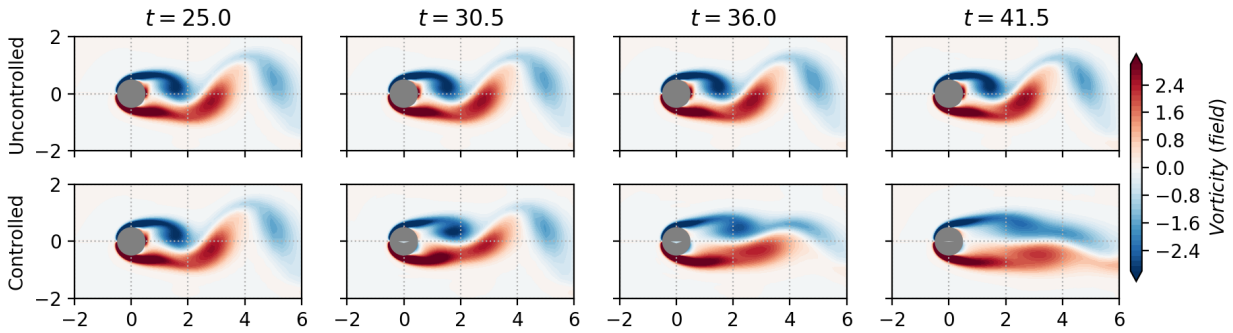


Figure 4.12: Comparison of uncontrolled (top) and controlled (bottom) flows in the transient phase of the control strategy.

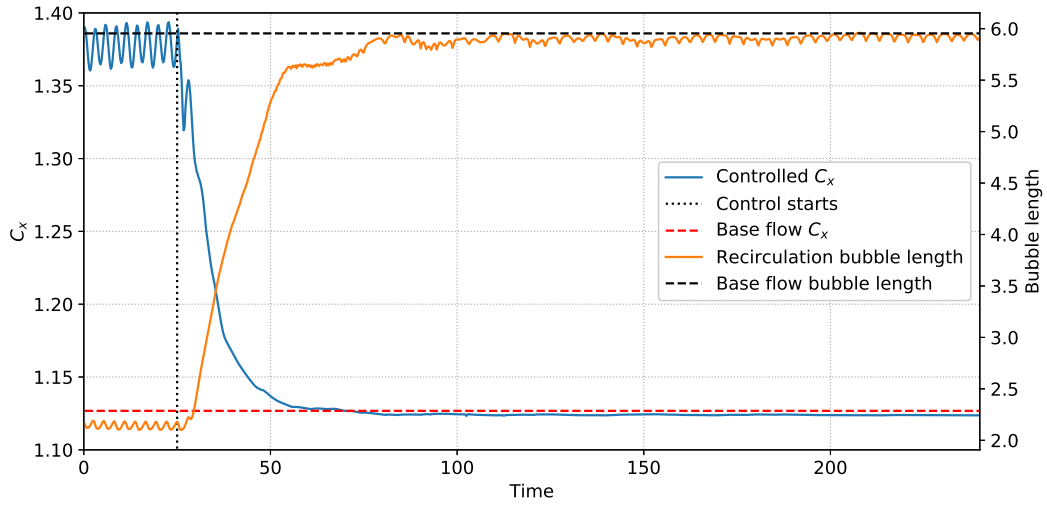


Figure 4.13: Evolution of C_d and of the length of the instantaneous recirculation bubble throughout time.

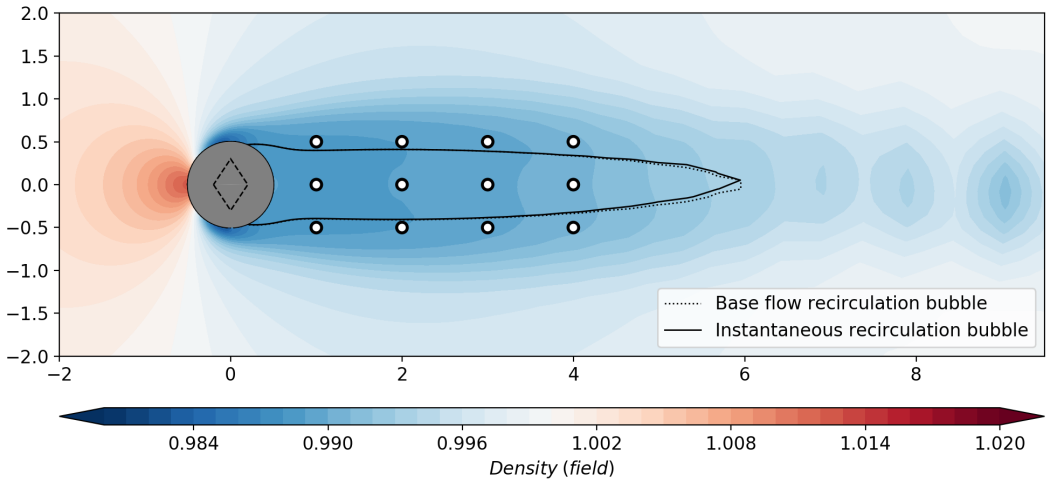


Figure 4.14: Comparison of the base flow and the controlled flow recirculation bubbles once the flow is stabilized.

As shown in figure 4.15 (left), the spectral analysis of the action during the second phase reveals two main oscillating components $St_1 \approx 0.11$ and $St_2 \approx 0.14$ and three secondary peaks at Strouhal numbers $\delta_{St} = St_2 - St_1$, $St_3 = St_1 + St_2$ and $St_4 = 2St_2$, the latter having amplitudes at least two orders of magnitude lower than the main components. Since δ_{S_t} and St_3 are the marks of nonlinear coupling between the two main components, one can assume a nearly interaction-free superimposition of the two main waves at St_1 and St_2. Their corresponding Fourier modes (not shown here) peak near the location of the "tail" of the recirculation bubble.

Note that in the stabilized phase, the state is close to the base flow and the actions are small, such that the flow evolves in a linear regime. The dominant Strouhal numbers of this phase are significantly lower than the natural vortex shedding frequency $St = 0.18$. This may be easily understood by performing a resolvent analysis, which describes the frequency-response of the flow in the vicinity of a steady state. If $\underline{q}$ denotes the flow state, $\underline{f}$ an external forcing, and $\mathcal{N}$ the Navier-Stokes operator, then recalling equation 2.2 introduced in section 2.1.5.4:

$$\hat{\underline{q}}' = \underbrace{\left(i\omega \underline{\underline{\mathbb{I}}} - \underline{\underline{A}}\right)^{-1}}_{\mathcal{R}} \hat{\underline{f}}, \tag{4.1}$$

with $\mathcal{R}$ being the resolvent operator. The highest singular value of $\mathcal{R}$, which is a function of ω, gives the highest linear gain σ that may be achieved through an external forcing (see for instance Beneddine [17]). Formally, it reads:

$$\sigma^2(\omega) = \max_{\hat{\underline{f}}} \frac{||\hat{\underline{q}}'||_{\underline{q}}}{||\hat{\underline{f}}||_{\underline{f}}}, \tag{4.2}$$

with $||\cdot||_{\underline{q}}$ and $||\cdot||_{\underline{f}}$ representing norms on the response and forcing spaces respectively (classically associated with the kinetic energy for the response, and the L_2-norm for the forcing). As illustrated by figure 4.15 (right), the highest optimal gain is obtained for $St = 0.12$ (consistently with Barkley [11], Jin *et al.* [107]) and the flow is responsive to only a narrow range of Strouhal numbers (below 0.15). It is therefore not surprising that the values associated with the control fall within this range. But interestingly, the control avoids the highest gain frequency and the particular selection of the

two specific frequencies $St_1 = 0.11$ and $St_2 = 0.14$ remains an open question. To our knowledge, this is not reminiscent of any existing work related to the linear control of the vortex shedding near the base flow. It is worth noting that both control time step and action interpolation have a negligible effect on the spectra since the control frequency is approximately 30 times larger than St_2. Thus, there is no effect of aliasing and no significant distortion due the interpolation method.

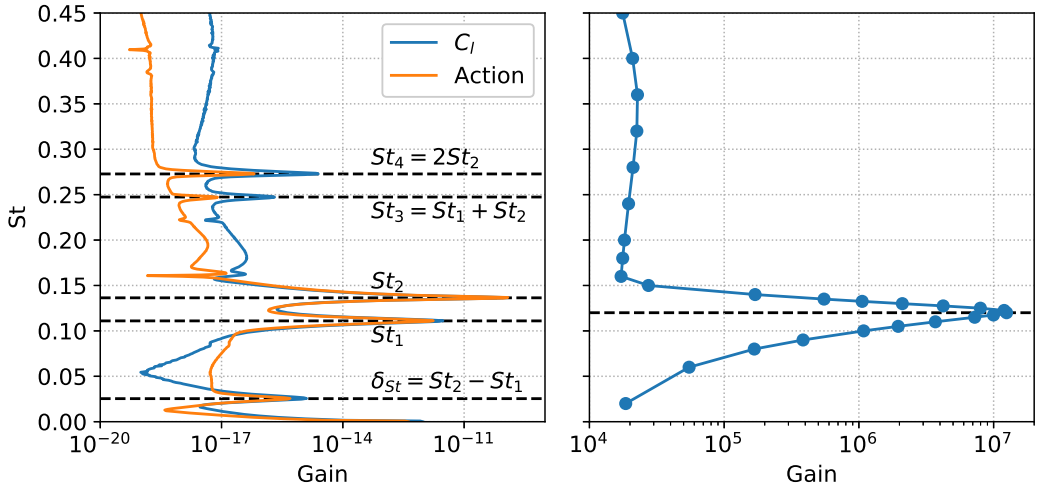


Figure 4.15: (left): Spectra of the lift coefficient C_l and of the action during the second control phase. (right): Evolution of the optimal gain of the base flow resolvent operator with the external forcing Strouhal number at $Re = 120$.

4.2.3 Robustness

As introduced previously, robustness is a key issue in RL since, contrary to linear control methods, no explicit robustness margins can be computed. Thanks to its trial-and-error paradigm, RL-trained agents inherit somehow the ability to recover from slightly sub-optimal control actions or biased measurements. Here, an empirical assessment of the policy robustness is performed on two aspects: a variation of the Reynolds number and additional measurement noise.

4.2.3.1 Variation in Reynolds

An assessment of the control policy robustness across a range of Reynolds numbers has been performed, using batches of 10 test cases having different random weight initialization. This issue is tackled in the study of Atam *et al.* [7], where gain-scheduled controllers are evaluated on a cylinder flow with time-varying Reynolds number. Unlike Tang *et al.* [237] who trained their policy on several Reynolds numbers values (100, 200, 300 and 400) and evaluated it on a mix of "seen" and "unseen" Reynolds numbers, our policy has been trained on a single Reynolds value $Re = 120$, evaluations have been performed on a range spanning from $Re = 100$ to 216 and compared with cases specifically trained on those Reynolds numbers. As illustrated by figure 3.12, this range of Reynolds numbers corresponds to a variation in vortex shedding Strouhal number of around 18%. Moreover, the non-dimensional amplitude of the pressure fluctuation displays a factor 2 between the two extreme Re values considered. This shows that the flow dynamics, although not radically

different, is still noticeably altered in this range of Reynolds number, such that the robustness is tested in actual off-design conditions.

Figure 4.16 shows that the control is remarkably robust. Note that, as previously introduced, the flow state is made non-dimensional by the reference density ρ_∞, upstream velocity U_∞ and static temperature T_∞. Reference velocities, pressures and case geometry (cylinder diameter and sensor location) being held constant is decisive for the robustness of the control policy. This ensures indeed a nearly constant convection time between sensors and comparable variation amplitudes both for sensors (on pressure) and for actuators (on mass flow) across the different Reynolds numbers considered. The only varying factor between different Reynolds flows is the change in vortex shape, their relative strength and organization. The policy, acting only as a function of the current observation s_t, is insensitive to the variation in the von Kármán vortex street convection velocity. It is hence only affected by the change in instantaneous form of the flow structures, and the present results prove that the control law handles these changes very well.

Non-dimensonalisation also circumvents the issue of neural network input normalization. Once neural networks' weights and biases are tuned to adapt to the range of input values, they remain appropriately tuned as this range does not overly change across Reynolds numbers. Tang *et al.* [237] used the same non-dimensionalisation scheme. Thus, even though their deep learning algorithm is different, the robustness they observed may be explained by the fact that the policy is robust over a wide range of Reynolds numbers even with a single Reynolds number training. Adding several other Reynolds numbers in the training marginally improves an already strong robustness.

Figure 4.16 also shows better robustness for lower Reynolds numbers than for higher ones (compared to the Re of training). One of the reasons may be that the chosen sensor layout, fixed across all cases, covers more of the base flow recirculation bubble for lower Reynolds numbers. It has been shown indeed that its length increases with the Reynolds number. The 12-sensor layout spans over 75% of the recirculation region at $Re = 100$, but only 55% at $Re = 216$.

4.2.3.2 Measurement noise

Assessing the tolerance of the control strategy to measurement errors is a key point in the transposition of that method to real-world experiments, where measurement noise is unavoidable. Noise robustness is therefore important in the perspective of transfer learning from a numerically trained case (without noise) to an experimental setup. To this end, the robustness of a zero-noise-training policy has been assessed and compared with policies trained on noisy data using batches of 20 randomly initialized test cases as previously done for the assessment of the Reynolds robustness. Added noise is parameterised, using a relative amplitude σ. Noisy observations $\tilde{s}_t$ are computed as:

$$\tilde{s}_t = s_t + \bar{s}_t \sigma \mathcal{N}(\cdot|0,1), \tag{4.3}$$

where $\bar{s}_t$ is the average pressure over all sensors at time t, which is found to be relatively steady and $\mathcal{N}(\cdot|0,1)$ is a standard random normal probability distribution. Figure 4.17 compares the performances of policies trained at different noise levels σ and evaluated on a range of noise levels from 0 to 1. One can notice that the level of training noise does not seem to impact performances in a significant manner up to $\sigma = 0.5$, which corresponds to very noisy measurements that certainly exceeds the actual noise one may expect in most experiments (see figure 4.18). Unexpectedly, figure 4.17 tends to show that a zero-training-noise policy seems overall slightly more robust to noise than others at different training noise levels. Therefore, in the present case, it is unnecessary to account for measurement noise during the training, which is promising for the possible transfer of CFD-trained models to experiments.

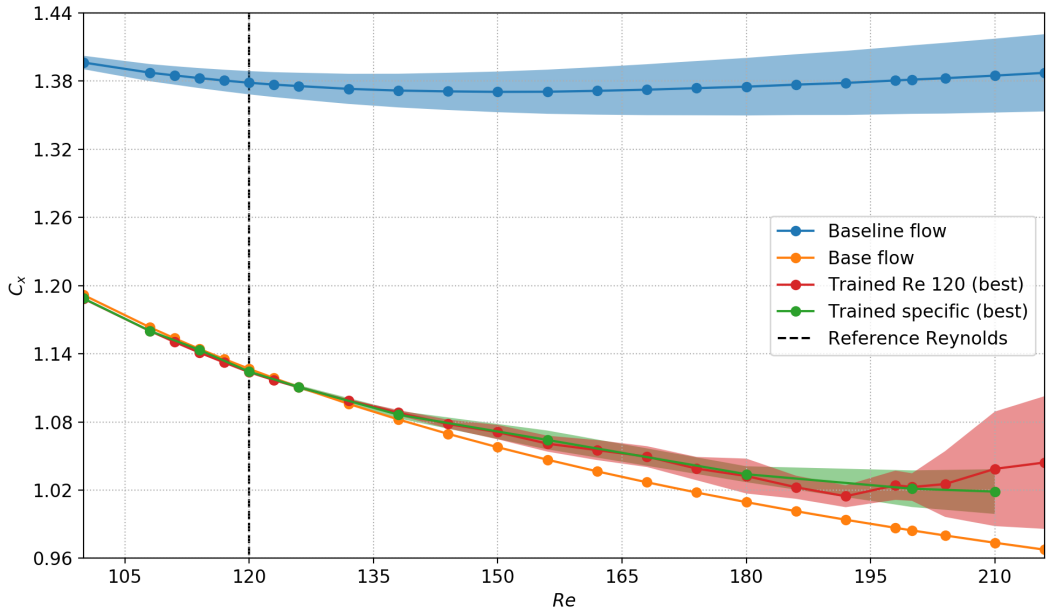


Figure 4.16: Robustness to a Reynolds number variation. The best case (among 10 test cases) trained at $Re = 120$ is tested at different Reynolds numbers (red curve). Its performance is compared to the best control policy (among 10 test cases) specifically trained at the test Reynolds number (green curve). Shaded areas represent the standard deviation of the controlled drag coefficient C_d.

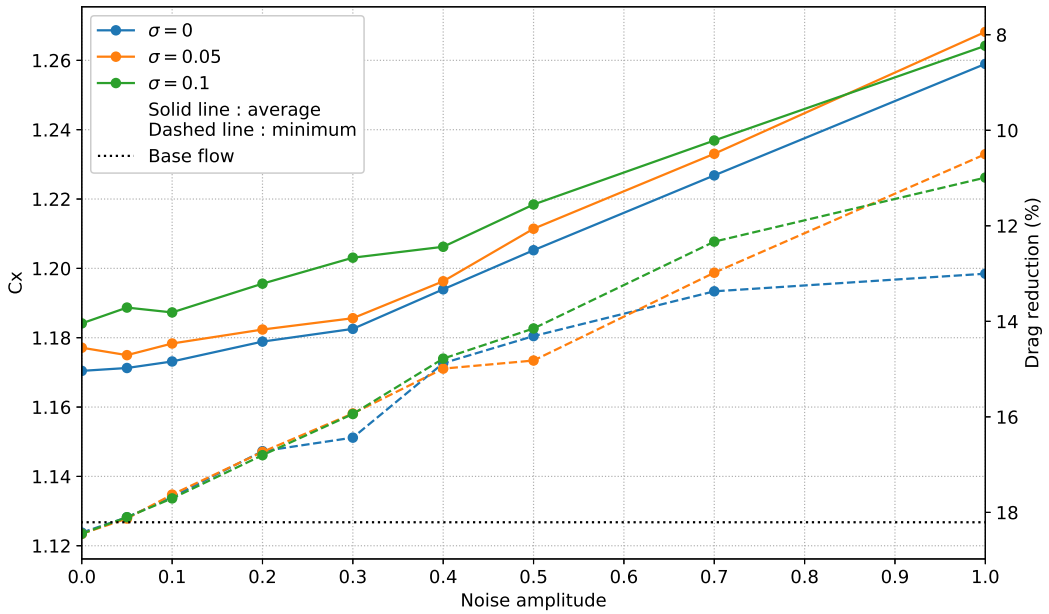


Figure 4.17: Robustness to Gaussian noise on observations. Each curve represents the mean (solid line) or the best (dashed line) performance in drag coefficient of a 20 test-case batch trained with noise levels from $\sigma = 0$ to $\sigma = 0.1$ and evaluated on noise levels ranging from $\sigma = 0$ to $\sigma = 1$.

Figure 4.18 illustrates this robustness throughout time for a policy trained with $\sigma = 0$. Despite large noise disturbances, the control policy achieves good performances. Even with extreme noise

levels such as $\sigma = 1$, the drag reduction reaches about 12% on average. This may be explained by the feedback characteristic of the problem that enables for efficient error correction from one control step to the next. Both observation and action signal-to-noise ratios (SNR) are assessed on

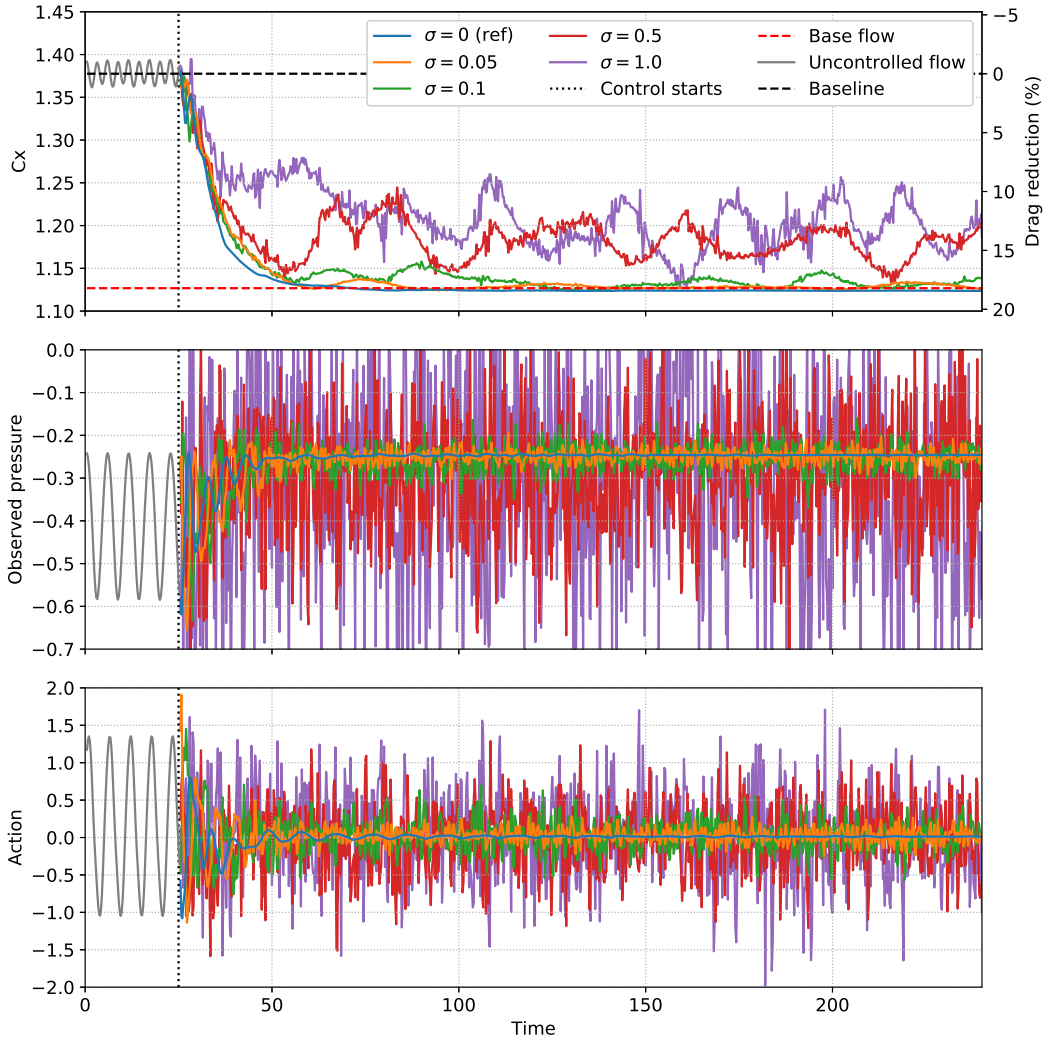


Figure 4.18: Robustness to Gaussian noise on observations for a policy trained without noise ($\sigma = 0$). (top): Evolution of C_d throughout time, for different noise levels. (middle): Noisy pressure signal s_0 located in (1,0.5). (bottom): Corresponding action taken by the actor.

the second control phase (having steady statistics) as:

$$SNR = \frac{\text{noise-free variation amplitude}}{\text{noise standard deviation}}. \tag{4.4}$$

Action noise is defined as the difference between the action computed using noisy observations and the action based on noise-free measurements. Results are reported in Table 4.3. Note that apparent discrepancy between SNR and σ values are due to the definition of each quantity: SNRs are computed considering the amplitude of variation of the signal, thus excluding the signal's time-averaged value, while the noise level driven by σ is measured as a fraction of the signal value, including its constant component. It can be seen that the action SNR has the same order of

magnitude as the observation SNR, their ratio ranges between 0.7 and 2. In particular, the SNR of observations and actions become closer as the level of noise increases. This highlights the robustness of the policy, which does not diverge from optimal actions due to spurious fluctuations within the observations. The errors do not accumulate over time and the closed-loop system seems able to rectify the previous erroneous action to contain the deviation from the optimal controlled flow-state. The policy is therefore sufficiently insensitive to input errors in that range of noise levels to ensure a strong robustness. In addition, it is possible that the decorrelation of these errors between each measurements helps mitigating the effects of the noise.

σ	Observation SNR	Action SNR	Average drag reduction (%)
0	∞	∞	18.4
0.05	0.33	0.17	18.1
0.1	0.18	0.09	17.8
0.5	0.04	0.04	14.2
1	0.02	0.03	13.1

Table 4.3: Noise robustness comparison. SNR: Signal-to-Noise Ratio

4.2.4 Impact of the episode and control step lengths

In the current section, the impact of the control step duration as well as the length training roll-outs are investigated on a slightly differently controlled cylinder flow. The observation setup is made of 9 sensors (in 3 columns) instead 12 sensors and the actuation is performed with 4°-wide jet inlets (instead of 6°).

As discussed by Rabault & Kuhnle [186], the choice of control step duration is driven by both flow and controller time horizons. The natural vortex shedding period imposes a higher bound on the control step for both measurements and actions to identify the dynamics without aliasing. 50 numerical steps (the duration Δt used in the previous study) correspond to approximately 22 control actions per vortex shedding period. The controller imposes a lower bound constraint since the impact of a given action a_t should not be sensed in a "too-distant" future, that is within a reasonable number (< 100) of control steps later. This effect is illustrated by figure 4.19, where the learning curves with different control step durations are compared. To keep all things otherwise equal, the episode length is adapted to the control step duration in order to span the same physical duration (or number of vortex sheddings). To keep the number of samples per epoch constant as well, the number of episodes run per epoch is also adapted.

As shown by figure 4.19 (left), below 5 steps per characteristic period training fails. This is simply due to aliasing, the agent cannot capture the system's dynamics properly and thus cannot "learn" proper control actions. With 5 and 20 steps per period, the learning performances appear maximized before degrading again with 50 steps per period. In the latter case, the system's response to forcing in a future further than 2 characteristic periods is more than 100 steps "away" for the agent. Thus these future rewards are weighed with an actuation coefficient $\gamma^n < 0.36$ with $n > 100$. For the current environment dynamics, the agent has been rendered "myopic", the relevant time horizon cannot be captured properly. At last, 5 steps per period seems to slightly over-perform 20 steps per period. Yet 20 steps per period is generally favored over 5 for the simple reason that this enables to produce four times more sampling data for the agent at a constant environment running cost. This is very valuable for costly environment such as CFD ones.

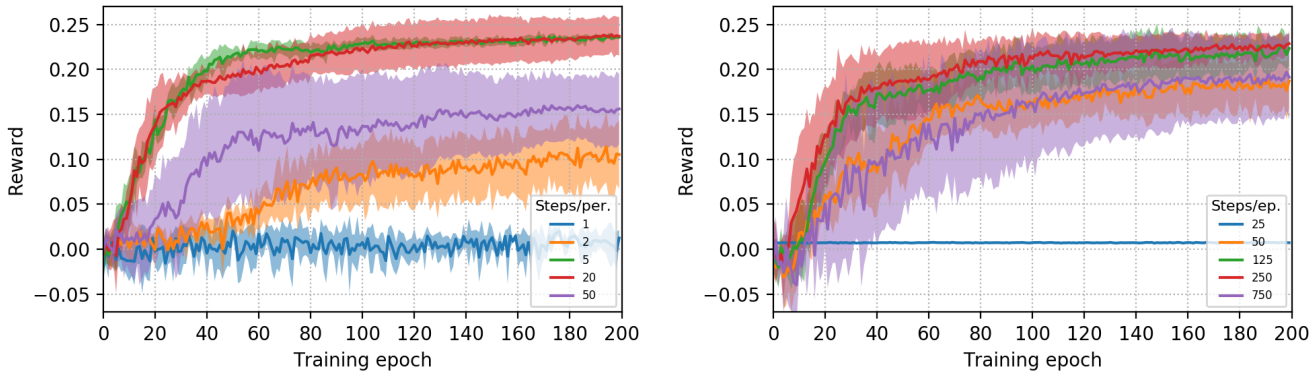


Figure 4.19: Comparison of training performances (average reward obtained on training runs) for (left) varying control step duration (measured as number of control step per characteristic period) and (right) roll-out length. In the latter case, the control step is set to its standard value ($\approx$ 22 control steps per characteristic period).

Figure 4.19 (right) illustrates the impact of the episode length. As previously, the study is led so that the total number of training samples is constant, thus the number of roll-outs per epoch is adapted. First and as expected here, one can notice that the longer the run is the better performances are. In the extreme case of only 25 control steps per episode, the performance remains close to the uncontrolled baseline. The agent is both constrained by the impossibility to collect samples quantifying the effects of control actions further than (at best) one vortex shedding period in the future. In that case, the average performance is computed only on the first steps where obviously the flow is not controlled and stabilized. For longer episodes the asymptotic performance stabilizes to larger values. That end-up converging to the reward of the controlled flow by effect of averaging over the whole episode.

Second, one can notice in the first epochs of training that the transient decrease in performance is of larger amplitude for longer episodes. This may be explained by the fact that initially random actions disturb the flow and the accumulation of these drives the flow further into disadvantageous states. On more chaotic flows, this phenomenon might be responsible for a strong limitation on the episode length and/or the exploration noise.

4.3 Controlling a stalled airfoil flow

These next paragraphs discuss the RL-based control of the low-Reynolds number stalled airfoil flow described in section 3.4.3.1. **Parts of this section are drawn from Paris *et al.* [175].**

4.3.1 Training and control performances

PPO-CMA has been run on test-cases with the previously introduced environment setup (refer to section 3.4.3.1) for angles of attack (α) of 12°, 15° and 20°. The choice of these three cases is motivated by the significantly different dynamical behavior they exhibit. Each epoch is composed of 16 runs of 250 control steps each (run on 5 parallel environments), which represents around 3.6 CPUh per epoch. For the sake of conciseness, comparison of the results between these cases is only presented and discussed when noticeable differences arise.

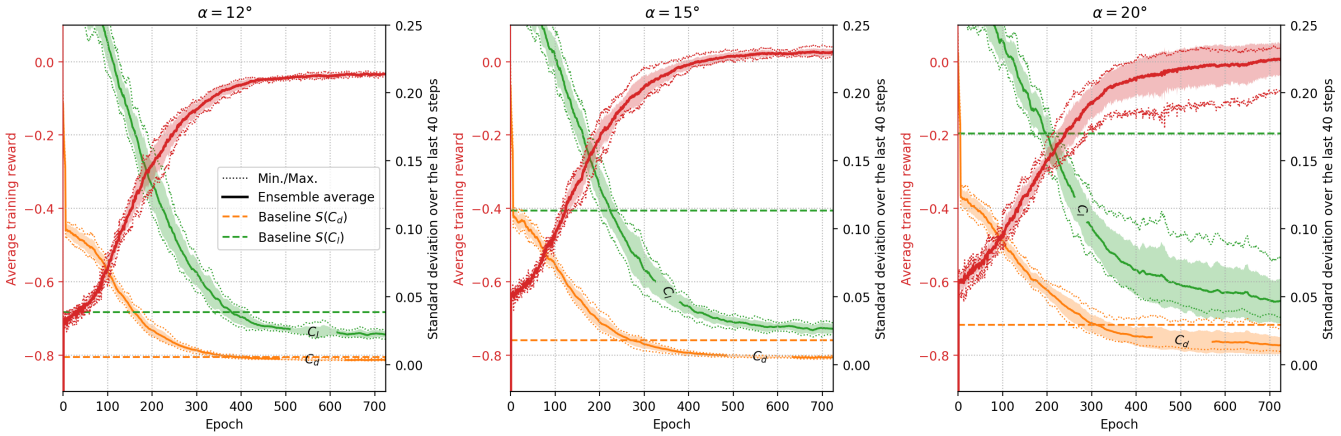


Figure 4.20: Ensemble-averaged learning curves of the pre-training phase for $\alpha = 12°$ (left), $\alpha = 15°$ (center) and $\alpha = 20°$ (right). Dotted lines describe the ensemble minimum and maximum and shaded areas illustrate the standard deviation across the batch.

For each angle of attack, ten independently initialized test cases have been trained. Figure 4.20 illustrates the evolution of both the average training reward and the standard deviation of the lift (C_l) and drag (C_d) coefficients at the end of each training run. During training and for all of the cases, the average reward grows rapidly until epochs $300 \sim 400$ then stabilizes to its maximum value. This comes with a steep reduction of the variations of both C_l and C_d. The average exploration component σ of the policy steadily decreases during training (not shown). This behavior is expected from the agent, that starts with a broad exploration of the possible actions and then narrows it down toward a more deterministic control strategy.

Figure 4.21 shows the ensemble averages of evaluations performed at the end of this pre-training phase for $\alpha = 15°$ and vorticity snapshots of the flow on a randomly selected test-run. As soon as the control starts, the control action swiftly drives the lift and drag coefficients toward favorable and stable values. As shown by both the extremal envelopes and the flow snapshots, this correlates with a stationarization of the flow through the cutoff of vortex shedding and with flow re-attachment. Control actions are moderate suction actions as expected[2]. In the following, actuators are numbered according to figure 3.13.

For $\alpha = 20°$, all action components display strong variations during the transient phase which lasts for about 8 non-dimensional time units. They all remain unsteady afterwards, synced on the vortex shedding that has not been entirely canceled. Action components 0 and, to a lesser extent 1, 2 and 3, display a strong suction control forcing after the transient. These forcings ensure the flow re-attachment. For cases with $\alpha = 15°$, actuators 3, 4 and 6 enforce fast varying actions. The re-attachment is again ensured by actuators 0 and 3, and to a lesser extent, actuator 1. It is worth noting that after the transient (lasting for around 6 non-dimensional time units), all the action components excepted actuator 6 become relatively stable. At last, for $\alpha = 12°$, only actuator 3 has a strong action variation during the transient of duration of around 5 non-dimensional time units. During the stabilized phase, action components 2, 3 and 7 display a strong and relatively steady suction forcing. Again here actuators 6 and 9 remain somehow unsteady compared to the others.

Long term dynamics of the controlled environment have been evaluated using batches of 10 to 20 evaluation runs of 2500 steps. Power spectral densities of both lift and drag signals (starting

[2]The left-hand side graph show the L_1 norm of the action, which always positive even for suction actions

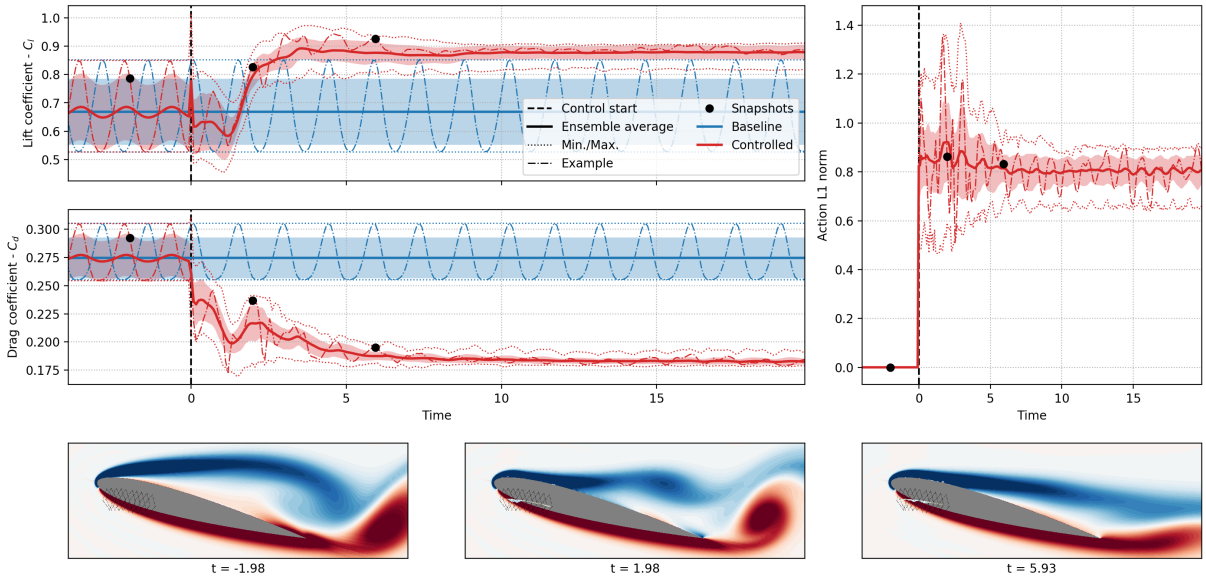


Figure 4.21: Ensemble averages of evaluation runs performed at the end of the training (epoch 700) for $\alpha = 15°$. Here 10 independent test-cases are evaluated on 10 test-runs (100 trajectories in total). Dotted lines describe the ensemble minimum and maximum and shaded areas illustrate the standard deviation across the batch. Vorticity snapshots illustrate the flow state at key moments of a randomly chosen run.

measurements after the control transient) have been estimated for each angle of attack and are reported in figure 4.22 in comparison of the previously introduced corresponding non-controlled signals.

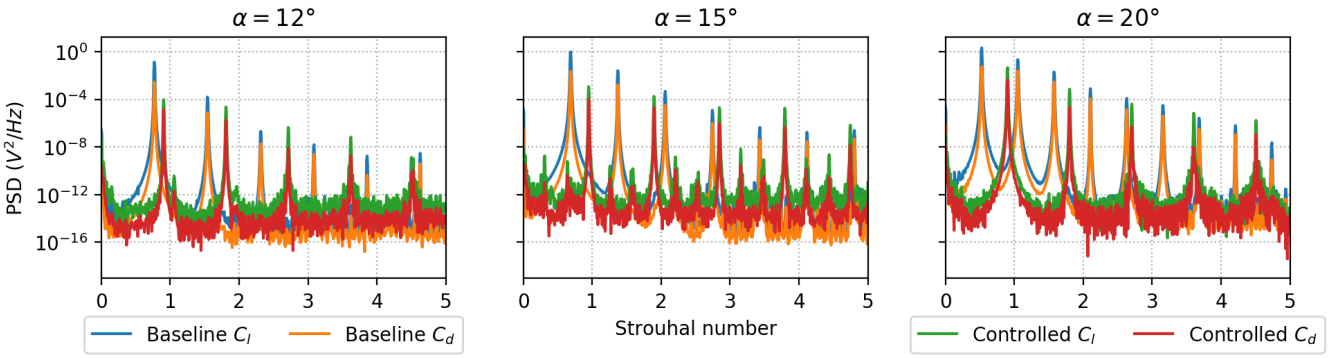


Figure 4.22: Comparison of power spectral density estimations of the lift and drag coefficients for $\alpha = 12°$ (left), $\alpha = 15°$ (center) and $\alpha = 20°$. The Strouhal number is computed as $S_t = f/f_{charact.}$ where f is current frequency and $f_{charact.}$ the reference frequency.

One can notice that for all studied angles of attack, control yields a shift of all the harmonics toward higher frequencies. This shift is around $0.15 \times n$, (n being the harmonic number) for $\alpha = 12°$, $0.25 \times n$ for $\alpha = 15°$ and $0.4 \times n$ for $\alpha = 20°$. This shift may be linked to the reduced height of the flow separation due to the (partial) reattachment yielded by control. By analogy with an unstable cylinder wake, one could assume a direct proportionality between the fundamental frequency of shedding and an equivalent height. Both C_l and C_d signal are amortized by a factor

12 (for $\alpha = 20°$) to 30 (for $\alpha = 12°$) for lift coefficient and by a factor 13 (for $\alpha = 20°$) to 200+ (for $\alpha = 12°$ and $\alpha = 15°$), confirming the nearly complete stationarization of the loads for these two angles of attack.

At last, the occurrence of pairs of sub-harmonics peaks for $\alpha = 15°$ may be noticed. They have not been related to a specific flow phenomenon or control behavior all the more that their amplitude is orders of magnitude lower than the highest peak of the spectra.

4.3.2 Closed-loop vs open-loop vs steady control

Even though the obtained policies are efficient considering the control objective, one can question the apparent complexity of these compared to the one of the training process. Where, for cases such as the cylinder flow, a precise syncing of the control action on the vortex shedding appears as a significant advantage for the performances of the optimized closed-loop control law compared to variable or constant open-loop controls, this may not be the case for the stalled airfoil flow.

Figure 4.23 compares the performances of RL-trained closed-loop policies with their open-loop equivalents. These are open-loop implementations of previously obtained control action sequences on differently reset flows or constant forcings corresponding to the final time-averaged control action.

One can first notice that, irrespective of the angle of attack α, all policies perform very similarly compared to the open-loop forcings. At $\alpha = 12°$ especially, a faster transient and stabilization using a constant control action can be noticed. For $\alpha = 15°$ and $\alpha = 20°$, closed-loop runs display a slightly higher performance but the edge on the open-loop forcings remains moderate.

Nb. act.	α	C_l	C_d	$S(C_l)$	$S(C_d)$	$S(C_l) + S(C_d)$
10	12	0.0%	0.0%	0.0%	0.0%	0.0%
	15	+2.4%	+1.7%	+98%	+9.9%	+75%
	20	+1.0%	+0.9%	+41%	+65%	+47%

Table 4.4: Comparison of the final performances (i.e. in the "stabilized phase") in closed-loop and open-loop conditions. Performance variations are measured as relative variation with respect to the closed-loop performance. Green colored figures indicate that open-loop control performs better on the metric than closed-loop whereas red colored ones states the opposite. Ensemble averages are computed on batches of 10 test runs for each to the 10 test cases (100 runs in total).

Table 4.4 reports the comparison between the closed-loop and the (variable) open-loop policy. Here percentages represent the relative variation observed on a given indicator of performance while switching from closed-loop to open-loop. These figures first confirm that variations in both lift and drag coefficients are marginal. Yet, concerning their temporal variation, computed as a standard deviation, the impact of the feedback observations directly influences the stabilization capabilities of the control, which is the primary objective of the training (the reward is defined on these fluctuations), the improvement of time averaged values of these coefficients being a "side-effect" of the flow reattachment.

One can then question the legitimacy of resorting to such complex and costly training processes if it is to "only" end-up with policies comparable to constant open-loop forcings in terms of performances on this case. One could justifiably argue than testing the performances of open-loop and/or "simpler" closed-loop methods could provide solutions as efficient as ours for a fraction of the computational cost.

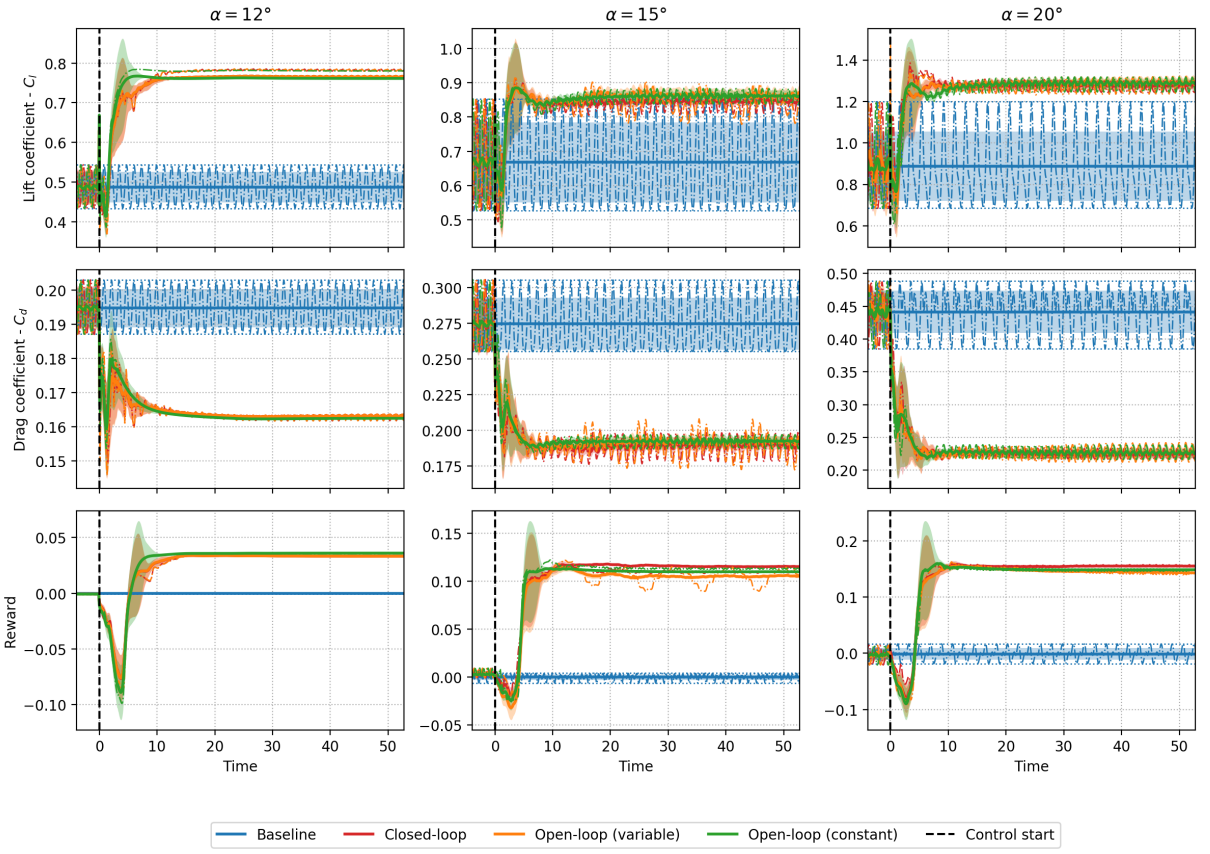


Figure 4.23: Comparison of the performances of the obtained closed-loop control policy (red line) with its open-loop equivalent (orange line, implementation of a sequence of actions drawn from another run irrespective of the current observation feedback) and with the constant open-loop control (green line) obtained by considering only the final time-averaged control action. Policies obtained at different angles of attack, $\alpha = 12°$ (left), $\alpha = 15°$ (center) and $\alpha = 20°$ (right) are compared regarding the lift coefficient C_l (top), the drag coefficient C_d (middle) and the reward r_t (bottom). Solid lines indicate ensemble averages across batches of 10 test-runs on 10 different policies (100 test-runs in total), shaded areas represent the ensemble standard deviation across this batch and dotted lines display a randomly picked example.

First and despite its apparent simplicity, a constant open-loop forcing still needs to optimize the value of 10 scalars (the value of each action component). This still requires a search of the action space (using gradient descent methods or gradient-free, evolutionary strategies). Second, it is empirically observed that RL-trained policies generally inherit a rather significant robustness to a variation in the flow conditions (refer to part 4.2.3), which may be interesting when applying these control methods to experimental setups where thermodynamics conditions are not perfectly controlled and measured. One can also point out, that these training strategies help discovering new control methods, that could not be uncovered using more traditional control synthesis ones. At last, despite bringing similar results, closed-loop policies still display a small performance edge over their open-loop counterparts.

4.4 Controlling an open-cavity flow

Controlling the open-cavity flow introduced in section 3.4.4 has been doubtlessly one of the toughest challenges of this thesis. This mainly comes down to the need for searching the hyper-parameter space in order to achieve efficient training. Some environments proved to be "tolerant" to a large range of hyper-parameter setups, enabling proficient control without having to browse the hyper-parameter space too much and test an extensive number of configurations. Concerning the present case, in contrast, this search has revealed to be a long and tenacity-testing endeavor.

4.4.1 LGP control

The first RL-based training tests dramatically failing to get even close to the non-controlled flow performance first led to consider LGP control strategies as a way to discover control strategies that RL was not capable of providing. As introduced, genetic programming has been experimentally implemented by Maceda *et al.* [149], who controlled the flow using plasma actuators and amplitude modulation. The current study uses a similar hyper-parameter setup but LGP policies that directly output the control action.

The tested setup presents only one actuator, implementing a Gaussian-shaped source term forcing centered in $(-0.1, 0.02)$ (directly above the upstream edge of the cavity) on the horizontal momentum equation, observations made of the last 3 horizontal velocity values sampled in $(0.9, 0)$ (in vicinity of the downstream edge of the cavity) and uses a reward considering only the standard deviation of the measurement signal across the last 100 control steps.

Parameter	Value
Population size	100
Allowed operations	$+$, $-$, $\times$, $/$, cos, sin and tanh
Number of variable registers	14
Maximum number of operations	100
Elitism	1
Cross-over probability	0.5
Mutation probability	0.4
Generation probability	0.1
Tournament fraction	7%
Tournament probability	1
Number of evaluation runs per individual	1
Control steps per test run	500

Table 4.5: Hyper-parameters of the LGP evolutionary process performed on the cavity test-case

Table 4.5 reports the hyper-parameters of the evolutionary search. Each individual is encoded by a sequence of at most 100 operations. Each operation is encoded, using the reverse Polish notation, as a matrix row, the top-down execution of operations/matrix rows implementing the control law. This may seam important but a large fraction of these operations lead to introns. 14 read-write slots are used to store input values or intermediate results. Individuals providing constant actions are rejected, i.e. not accepted as valid ones in the evolution process and thus re-sampled. New individuals are compared to already existing ones and rejected if their L_2 similarity on a set of 100 randomly selected inputs is above a given threshold with any of the other already seen individuals.

The first phase of a population evolution consists in performing elitism and the tournament phase. Here the tournament consists in extracting 100 individuals (with possible repeats) by iteratively sampling 7% of the old population and select to the best one with probability $p_{tour} = 1$. In the case where the tournament probability p_{tour} is lower, the second best is selected with probability $(1-p_{tour})p_{tour}$, the third best with probability $(1-p_{tour})^2 p_{tour}$ and so on and so forth. The obtained population then goes through cross-over, mutation or random generation. Cross-over cases where both individuals are identical (because of potential repeats) are re-sampled.

Figure 4.24 (left) illustrates the evolution of the performance of the population throughout the training stopped at epoch 20. One can first notice that the best performance (the lower the better in that case) is very quickly reached and only marginally improved afterwards. Second the population spreads over a wide range of performances, most of the individuals performing significantly worse than the non-controlled flow (which has an average performance of 0). Both right-hand side graphs of figure 4.24 describe the performance of the best individual, which has the following analytical expression:

$$a_t = \tanh\left(-0.23197 \times \tanh \circ \cos \circ \tanh \circ \tanh \circ \cos(2.10256 + s_t)\right),$$

where s_t is the most recent observed x velocity at $(0.9, 0)$ and a_t is the control action. It can be noticed that despite providing a somehow satisfactory short-term control (at least concerning the observed signal), long-term dynamics display a destabilization of the flow. This may be due to a slow unstable mode (potentially linked to the recirculation in the cavity) developing on time scales much longer than the evaluation run. As shown by the bottom right-hand side graph, the control policy allows for doubling the frequency of the control action with respect to the one of the observations if these oscillate in the range $[12, 23]$. This may be an important characteristics to the relative performance of the control.

An important factor limiting the efficiency of the LGP on such cases is the difficulty to converge low-variance, unbiased estimates of the fitness of individuals, in other words the need for an important number of (very) long test-runs to assess the value of the individuals. And as the vast majority of the individuals performs very poorly, it can be considered as a large waste of computational time. Considering the computational cost of these 20 epochs, i.e. running 2000 runs of 500 steps, and a critical lack of reproducibility or at least weak guarantees of performance, it has been decided to further study more complex RL-based trained strategies. Future developments could consider running a few epochs of LGP and reuse (by transfer learning) the best individuals as "seeds" for RL agents, ensuring a "flying start" and thus a reduced convergence duration.

4.4.2 Distillation between RL agents

As training a RL-agent using observations located near the downstream edge of the cavity proved to be notoriously challenging (likely because of the convective delay between control actions and consequences on observations), this test case has been chosen to test policy distillation. As previously introduced, policy distillation consists in transferring knowledge (by imitation) from an expert agent, here trained in an easier context, to a student agent, tasked with reproducing the expert behavior but here with a different set of observations. The principle is illustrated by figure 4.25 while both observation setups are reported in figure 4.26.

Figure 4.27 illustrates the impact of policy distillation on the performances of agents having a "student" observation setup. One can notice a sharp increase of the performances during the training phase where distillation occurs. Yet, it plateaus afterwards and the student performance

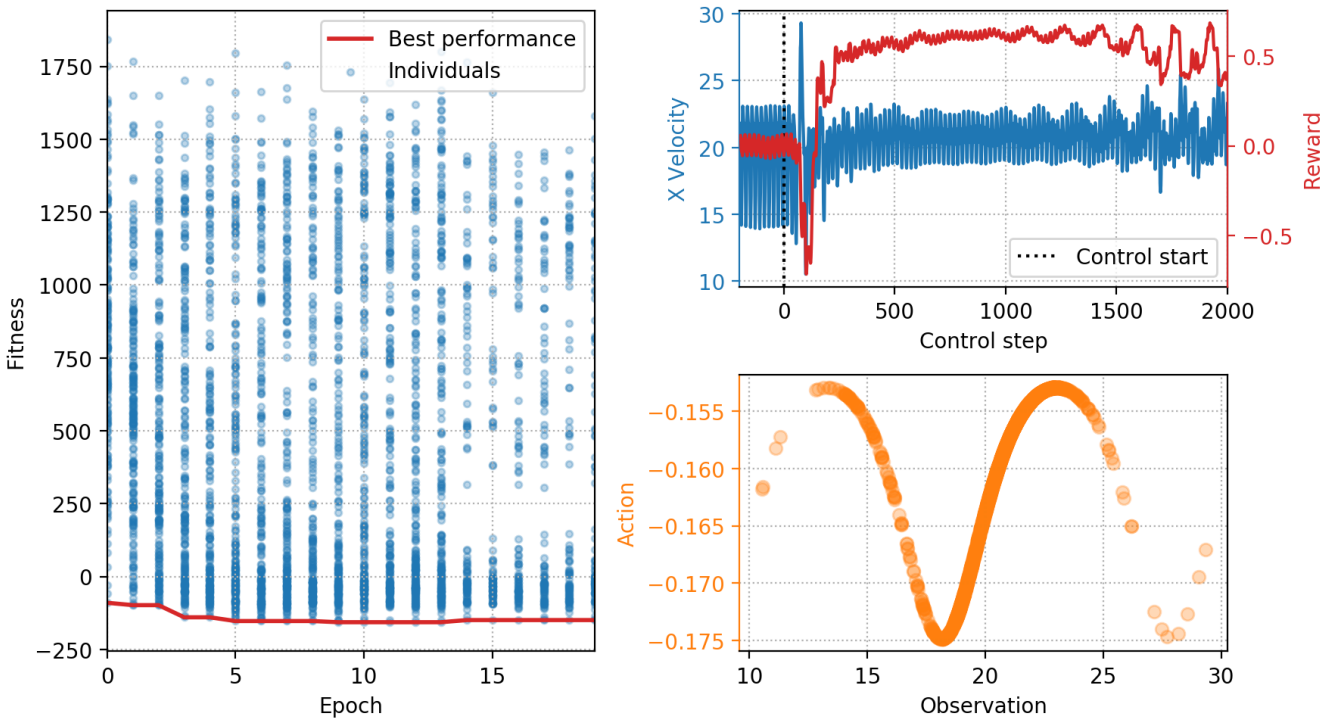


Figure 4.24: (left) Evolution of the performances of the population. (top right) Evolution of the x velocity observation on a test run of the best individual. (bottom right) Control law encoded by the best individual.

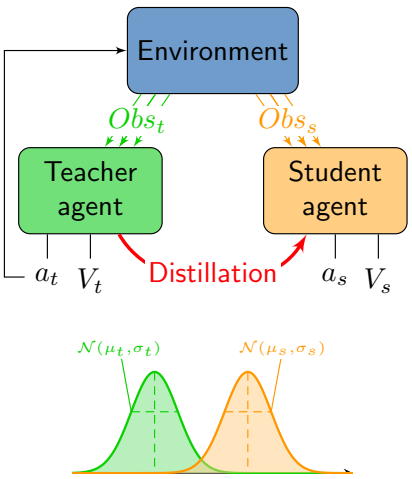


Figure 4.25: Schematics of policy distillation in an Expectation-Maximization (EM) context.

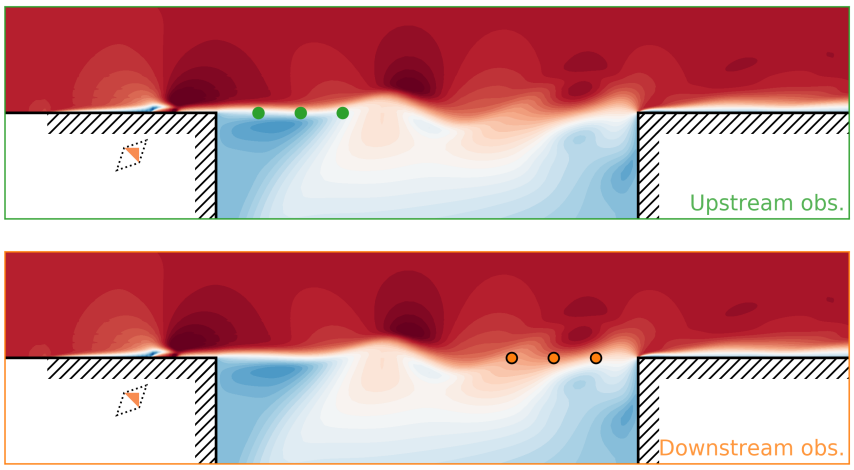


Figure 4.26: Comparison of both observation setups. (top) Expert agent setup with upstream observations. (bottom) Student agent setup with downstream observations.

remains significantly lower than the one of the expert (dashed red line in the figure). This gap can be expected from the fact that perfect imitation cannot be guaranteed simply because both observation setups are not equivalent from an informational viewpoint. Longer distillation phases have been testing but lead to a drop in the performance of the student. Still, this lack of performance

improvement calls from more investigations on this knowledge transfer strategy.

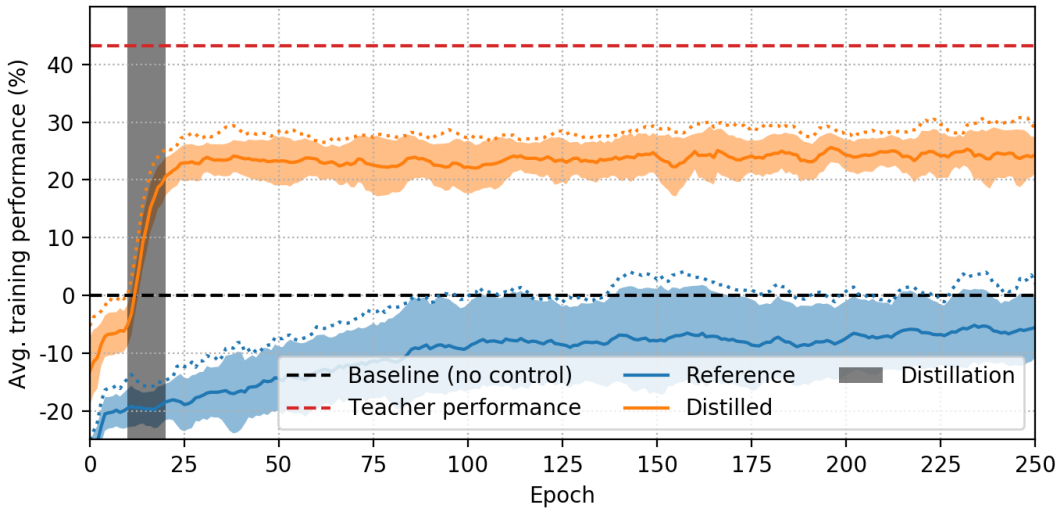


Figure 4.27: Comparison of learning curves of agents having the "student" observation setup, with and without distillation. The grayed out area represents the epochs where distillation from the expert has been performed in addition to standard training.

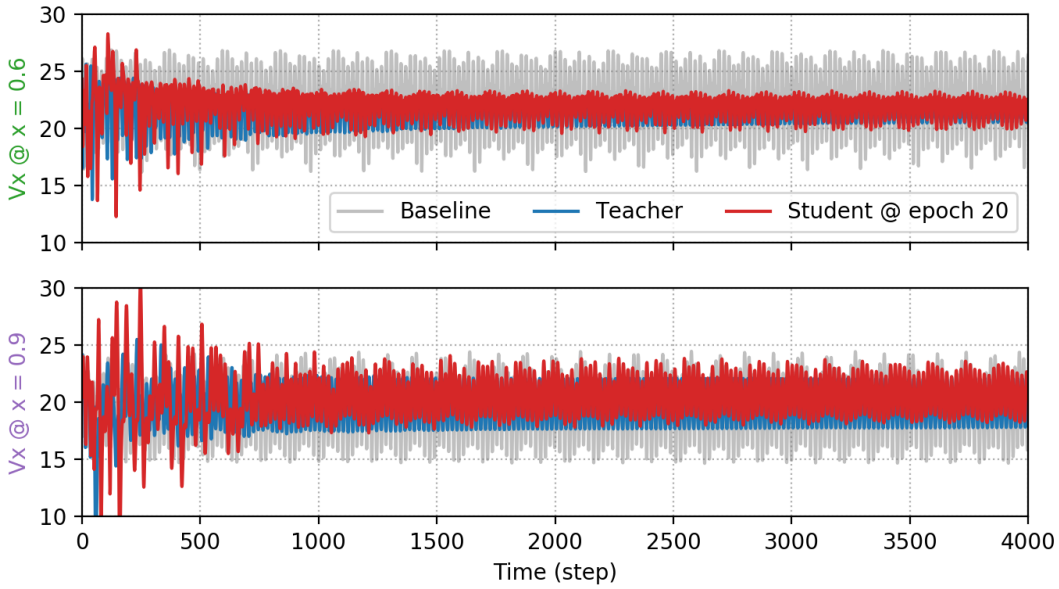


Figure 4.28: Comparison of both teacher (blue line) and student (red line) policies with the non-controlled signals (light gray line) at two different locations ($x = 0.6$ and $x = 0.9$ over the cavity) on long-term horizons.

Both performances are compared in long-term horizons on figure 4.28. As a training episode lasts for 500 control steps in the current study, the behavior observed from control step 1000 onward is likely a generalization of the behavior learned on earlier observations corresponding to a less stabilized flow. First one can notice that, contrary to the LGP control attempt, both cases (expert or teacher) do not see the emergence of a slowly growing destabilization (at least within

this time horizon). This may be thanks to the empirical robustness of RL-based control policies. Second, the imperfect knowledge transfer from the teacher to the student agent is confirmed by inferior performances materialized by a worse stabilization of the flow from the student (red line) than from the teacher (blue line).

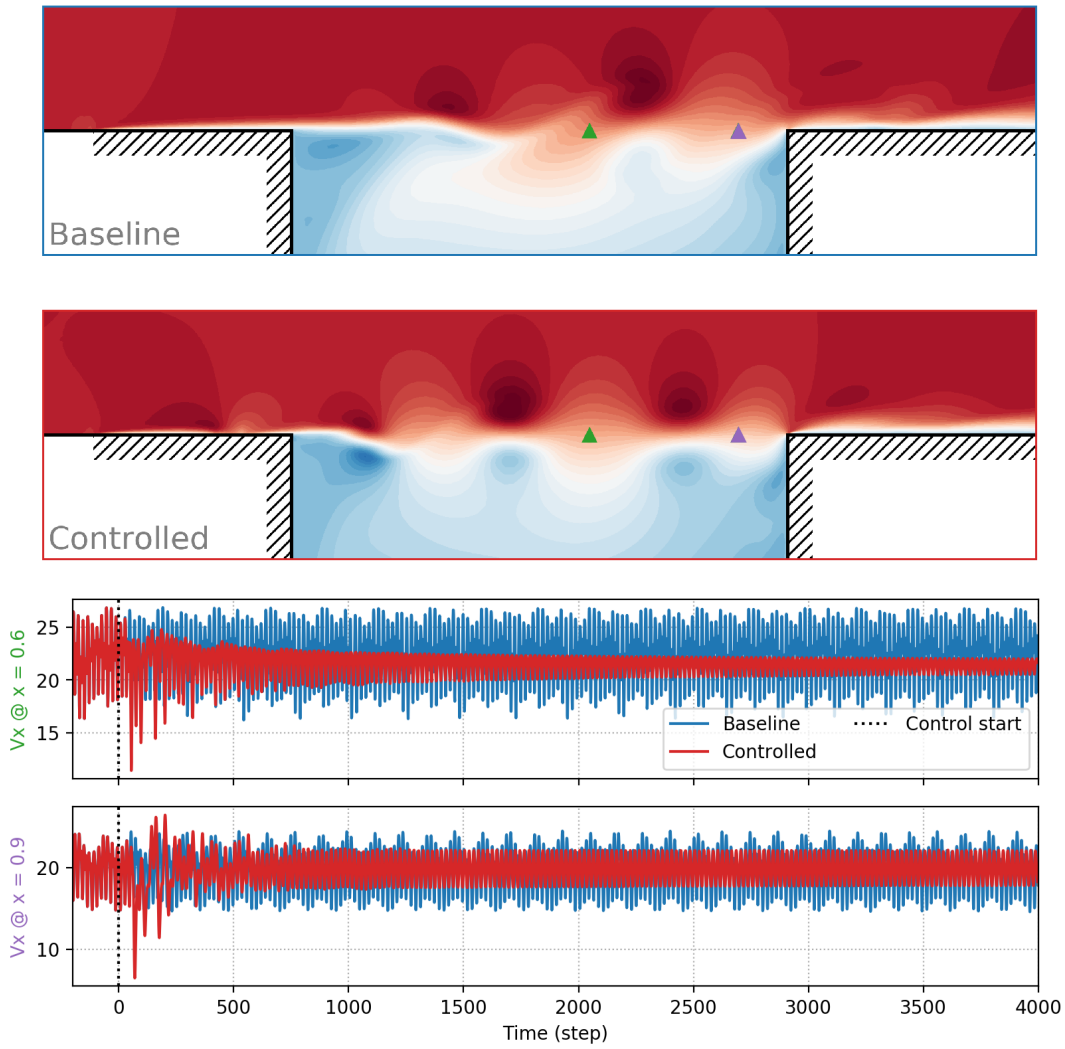


Figure 4.29: Comparison of the control performances of the policy trained with upstream sensors. (top graphs) Uncontrolled baseline (blue-framed) and controlled (red-framed) flow snapshots of the x-velocity at control step 4000. (bottom graphs) x-velocity signals at $(0.6, 0)$ and $(0.9, 0)$.

In that case, policy distillation provides encouraging results, calling for more studies on that strategy. Yet, when showing controlled flow snapshots, it has been realized that, contrary to what the horizontal velocity observation implies (i.e. a reduction of the flow unsteadiness), the overall fluctuation level was not reduced. Instead, taking advantage of the permissive formulation of the reward (only considering one measurement location), the agent had learned a strategy consisting in alternatively shedding vortices above and below the $y = 0$ line (refer to figure 4.29), on which the observation is located, so that the velocity signal was effectively reduced in contrast to nearby zones of the flow where vortices had been deflected.

The following part discusses the strategy developed and the attempts made to really control the

flow and drive it toward a thorough steadiness.

4.4.3 Reaching complete stabilization

The following RL-based study adopts the control setup described thereafter. **This section is drawn from an article to be submitted to JFM.**

4.4.3.1 Control and training setups

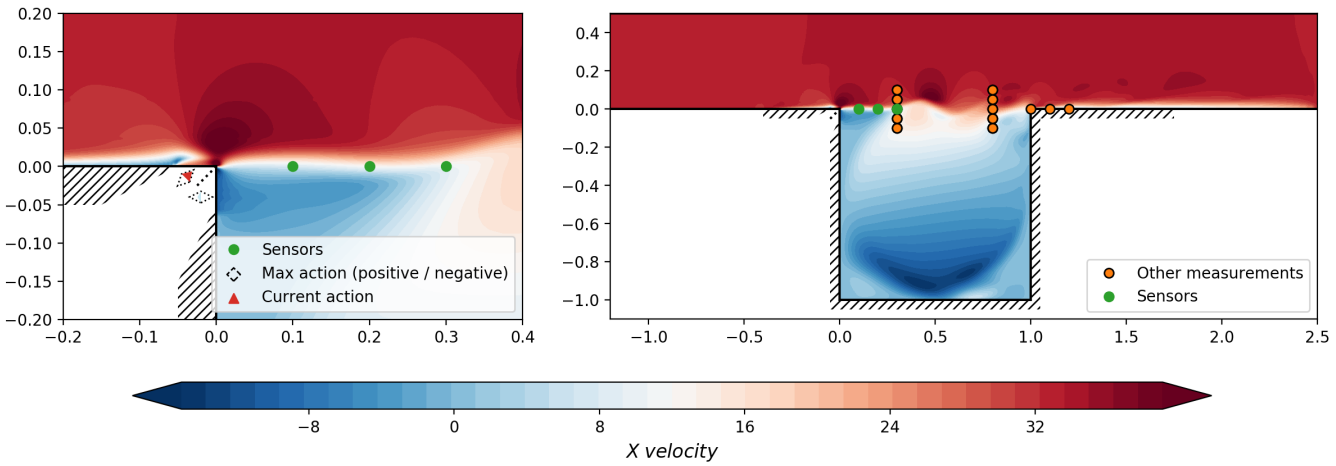


Figure 4.30: Schematics of the control layout. (left) Close-up on the upstream cavity edge. Blowing/suction control actions are symbolized by the colored triangles whose color and height depict the current control action. Dotted diamond represent the extreme forcing actions, ranging in $[-1; 1]$. (right) Full view of the cavity. Pressure sensors are represented by green dots whereas other measurements, used to compute multiple performance metrics are shown in orange.

Similarly to standard linear closed-loop design, the flow is considered as a plant stepped forward in time in a discrete fashion. The control step Δt is defined as the number of numerical iterations ran in-between two measurements and/or control action updates. In the current setup, $\Delta t = 30\,\delta t = 8.22 \times 10^{-4}$ non-dimensional time units. Thus, it takes approximately 20 control steps to run one characteristic period. Such a value enables to both avoid aliasing measured signals and allows for having a few periods within a medium-term future (< 100 control steps), as discussed by Rabault & Kuhnle [186]. The current state of the flow is partially observed through 3 pressure measurements from virtual probes located in $(0.1, 0)$, $(0.2, 0)$ and $(0.3, 0)$, as shown by figure 4.30 (right), slightly downstream the first edge of the cavity. The reference inflow static pressure p_∞ is subtracted to these signals to make up the observed partial state s_t. The control action a_t is implemented using two blowing/suction actuators. As illustrated by figure 4.30 (left), a first actuator is simulated on the horizontal wall upstream of the edge and spans $x \in [-0.05, 0]$. This actuator can either blow a jet at a downstream angle of 45° (positive action) or suck fluid (negative action) and is referred to as "top action" in the following. The second actuator, referred to as "wall action", is implemented on the neighboring vertical wall, spans $y \in [-0.05, 0]$ and blows horizontally or can similarly suck mass-flow. The idea behind this action layout is to provide the controller with the ability to significantly alter the flow in a zone where fluctuations start developing. A control action then ranges $[-1; 1]^2$ and is converted as a command in mass-flow rate at the previously described

boundary conditions. Similarly to the other introduced test-cases, and to avoid discontinuous and nonphysical changes in the boundary conditions simulating the action, linear interpolation ramps covering 90% of the control step, smooth out the control action for one control step to the next. Thus, at the i^{th} numerical iteration of control step t, the mass flow per unit area q_i imposed at the blowing/suction boundary conditions, is measured as a fraction of the free-stream mass flow rate and reads:

$$q_i = \rho_\infty U_\infty \left(a_{t-1}(1 - r_i) + a_t r_i\right) \text{ with } r_i = \begin{cases} i/27 & \text{if } i < 27 \\ 1 & \text{otherwise,} \end{cases}$$

The total enthalpy h imposed for the blowing control actions is the free-stream one, h_∞.

Other measurements are performed, alongside the partial observations previously introduced, at the end of every control step. Two groups of virtual probes measure the pressure and both components of the velocity, respectively at locations $\{(0.3, y_{probe}) \,|\, y_{probe} \in [-0.05, -0.025, 0, 0.025, -0.05]\}$ for the first upstream group of the probes and $\{(0.8, y_{probe}) \,|\, y_{probe} \in [-0.05, -0.025, 0, 0.025, -0.05]\}$ for the second, more downstream one. These probes buffer the last 40 measurements (corresponding to ≈ 2 characteristic periods). From these one can define two performance metrics m_3 and m_8 quantifying the stabilization of the flow:

$$m_3(t) = \frac{1}{5} \sum_{y_{probe}} S\left(p_{(0.3,y_{probe})}, t\right) + 0.1 \times S\left(vx_{(0.3,y_{probe})}, t\right) + 0.1 \times S\left(vy_{(0.3,y_{probe})}, t\right)$$

$$m_8(t) = \frac{1}{5} \sum_{y_{probe}} S\left(p_{(0.8,y_{probe})}, t\right) + 0.1 \times S\left(vx_{(0.8,y_{probe})}, t\right) + 0.1 \times S\left(vy_{(0.8,y_{probe})}, t\right),$$

where $S(\cdot, t)$ is the standard deviation of the signal recorded in the corresponding buffer at control step t, i.e. the fluctuations of the signal over the last ≈ 2 periods. As the shear layer develops with the convection of the fluctuations, the downstream-most metric m_8 is likely to be more faithful to a global stabilization of the flow. Yet, for reasons discussed in section 4.4.3.5 and unless otherwise stated, the reward r_t is defined as:

$$r_t = -m_3(t) - 0.1 \times \overline{p_{(0.3,0)}} - 0.1 \times ||\overline{a_t}||_1,$$

where $m_3(t)$ is the value of metric m_3 at control step t, $\overline{p_{(0.3,0)}}$ is the average pressure measured over buffer $p_{(0.3,0)}$ and $||\overline{a_t}||_1$ is the average control action bias over the last 40 control steps. Thus, maximizing the reward is equivalent to minimizing the fluctuations measured by m_3, while both keeping a pressure close to the one of the reference one and a forcing action with minimized continuous components.

In the current study, each epoch contains 3 roll-outs of 250 control steps each. To take advantage of the high-performance computing cluster used to train the cases, these roll-outs are performed in a parallel fashion, leveraging the MPI standard for the parallelization. Three environments are thus simulated on "client" ranks and the agent is handled by a "'main rank". Environment ranks then query the agent rank for control actions, step forward in time, send the observations and the reward back and iterate these operations until the end of the roll-out before resetting their state with probability $P(\text{env. reset} \,|\, \text{epoch})$. Unless otherwise stated, the following scheduling is implemented:

$$P(\text{env. reset} \,|\, \text{epoch}) = 1 \times (1 - f(\text{epoch})) + \frac{1}{20} \times f(\text{epoch}),$$

$$\text{where } f(\text{epoch}) = \max\left(0, \min\left(1, \frac{\text{epoch} - 300}{500 - 300}\right)\right),$$

is a ramp function between training epochs 300 and 500. One can then compute the expected number of control steps ran in-between two environment resets, that grows from 250 steps before epoch 300 to 5000 steps for epoch 500 and onward, as illustrated by figure 4.31 (left). Refer to section 7.1.2.2 for a more complete discussion of this scheduling. A reset of the flow state consists in over-writing it with a fully developed, non-controlled flow. The flow is then stepped forward for a random number of numerical iterations and all data buffers are re-initialized.

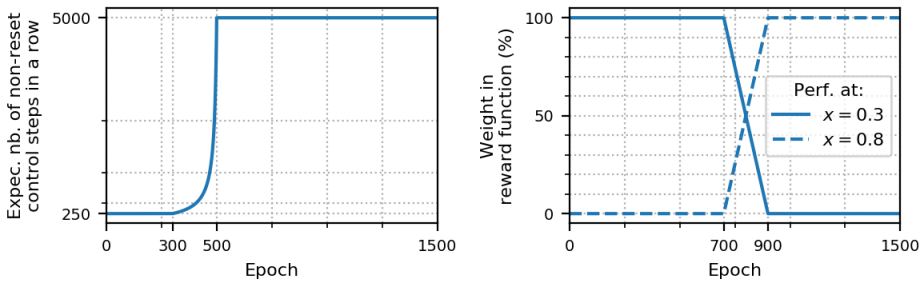


Figure 4.31: (left) Environment reset scheduling used in this study. Expected number of control steps ran in a row on an environment in-between two resets with respect to the training epoch. (right) Scheduling on the reward definition between the two metrics m_3 and m_8, discussed in section 4.4.3.5. This scheduling is **not implemented in the reference training setup**.

4.4.3.2 Control performances

A batch of 20 test-cases having the previously introduced setup has been trained on 1500 epochs. Control effectiveness is assessed via both performance metrics m_3 and m_8 previously introduced. Unless otherwise stated, these are normalized by the average performance of the non-controlled flow ($m_{3,\text{baseline}}$ and $m_{8,\text{baseline}}$ respectively):

$$m_3 \leftarrow \frac{m_{3,\text{baseline}} - m_3}{m_{3,\text{baseline}}}$$
$$m_8 \leftarrow \frac{m_{8,\text{baseline}} - m_8}{m_{8,\text{baseline}}}.$$

Thus, in the following m_3 and m_8 range $[-\infty, 1]$, where 1 corresponds to a perfect steadiness and 0 to the same level of unsteadiness as the non-controlled baseline.

Learning curves of these two indicators are displayed on figure 4.32. One can notice a drastic improvement of both normalized performance metrics m_3 and m_8 in the first 300 epochs, then a slower increase, once the agents perform better than the non-controlled baseline. As expected, both metrics are correlated.

One may point out the discrepancy in performance between the evaluation runs (of length 4000 control steps) and the training runs (of length 250). Two main factors account for this fact. First, during the training run, the control is sampled on $\mathcal{N}(\mu(s_t), \sigma(s_t))$ whereas evaluation runs only consider the best action $\mu(s_t)$. The training control action is thus overall more noisy than for the evaluation. On a setup where the goal is to damp fluctuations, noisy control actions obviously contribute to lower the performance. Second, training roll-outs may be performed on environments that may have been reset more or less recently in the past, thanks to the scheduling on the probability $P(\text{env. reset} \,|\, \text{epoch})$, contrary to evaluation runs all starting from a fully developed flow.

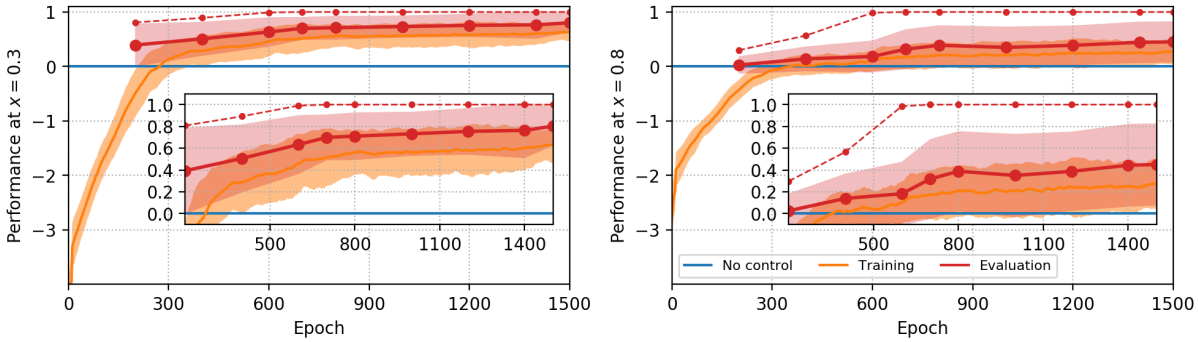


Figure 4.32: Learning curves of the final value of the performance metric (respectively m_3 and m_8) on training roll-outs of length 250 steps (orange lines) and on evaluation roll-outs of length 4000 steps (red lines). These performances are normalized by the average performance of the non-controlled baseline. Solid lines represent the ensemble average across the batch, while shaded areas illustrate the ensemble standard deviation and dashed lines represent the maximum value.

From epoch 600 onward, the maximum achievable performance is reached on evaluations for at least one test-case. Yet, the ensemble averages show lower performances indicating that all agents do not converge as well as the best individual. Figure 4.33 accounts for this by clustering the agents' performances into three different groups. The first and best-performing one is made of 5 test-cases, whose final normalized metric value m_8 is larger than 0.8. They all display an overall suction for the top action and strong blowing for the wall action. The second group (in orange on the figure) is made of 7 individuals and display performances that stabilize around 0.5 for all the indicators. Most of these converge toward blowing control actions for both components. For the majority of these cases, the performance tends to stabilize in the long run. At last, a group of 8 individuals show performances than are on average as good as the non-controlled baseline. They generally use a moderate blowing for the top action (not shown on the figure, hidden behind the previous group) and a small blowing for the wall action.

This shows that, contrary to other flow environments, results obtained on this cavity flow suffer from a lack of reproducibility. Multiple factors can account for this, as discussed in the conclusions.

4.4.3.3 Physical analysis

For the following sections, the best agent obtained on the 20-test-case batch is considered. Figure 4.34 illustrates the evolution of multiple performance indicators as well as flow snapshots at different moments of the evaluation run. The control actions appear to a have rapid stabilizing effect on the velocity measured in the shear layer near the upstream edge. It increases quickly within the first 50 control steps and then stabilizes. Conversely, the velocity measured at the bottom of the cavity (refer to the center graph) displays a much slower stabilization. This fact is confirmed by the center flow snapshot at step 250 that shows remaining convective structures in the recirculating flow of the cavity whereas the shear layer seems already totally stabilized.

This lag weighs on the value of performance metric m_8, likely thanks to the pressure probes that are exposed to the radiated acoustics from these recirculating fluctuations. Thus, even if most of the performance is gained in the first 200 control steps, it takes around 1000 steps to thoroughly stabilize the flow. The control action, appears to fluctuate noticeably at the beginning of the run but then slowly stabilizes toward non-null averages. The role of these fluctuations is further discussed

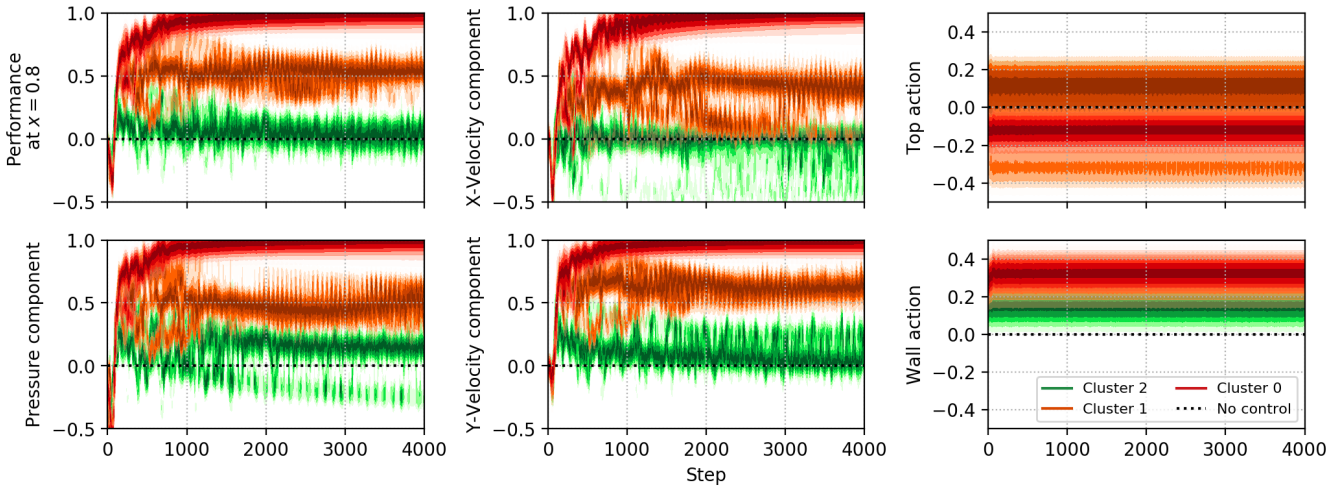


Figure 4.33: Heat map of the performances of the agents of the test-batch on evaluation runs. They are clustered into three groups depending on the final value of the normalized performance metric m_8: red if it is larger than 0.8, green if it is lower than 0.25 and orange in-between. (top left) Normalized performance metric m_8. (bottom left, top center and bottom center) Pressure, horizontal velocity and vertical velocity components of m_8 respectively. (top right and bottom right) Control action, top and wall action respectively.

in section 4.4.3.6.

To analyze in more details the control action, a global stability is performed. Figure 4.35 compares the absolute stability of the obtained steady controlled flow with the base-flow (steady solution of the Navier-Stokes solution) of the non-controlled flow. Global absolute stability is assessed as follows. Let $\underline{q}$ and $\mathcal{N}$ respectively denote the flow state (full-state) and the Navier-Stokes operator. One can then write:

$$\frac{\partial \underline{q}}{\partial t} = \mathcal{N}(\underline{q}).$$

If $\underline{q}$ is later decomposed as the sum of a steady state $\underline{q}_{BF}$, solution of the Navier-Stokes equations (i.e. $\mathcal{N}\left(\underline{q}_{BF}\right) = 0$) and a small perturbation vector $\underline{q}'$, one can develop a first order Taylor expansion as:

$$\frac{\partial \underline{q}'}{\partial t} = \underline{\underline{J}}.\underline{q}' + \mathrm{o}(||\underline{q}'||) \quad \text{where } \underline{\underline{J}} = \left.\frac{\partial \mathcal{N}}{\partial \underline{q}}\right|_{\underline{q}_{BF}}.$$

One can study the spectrum of matrix $\underline{\underline{J}}$, referred to as the Jacobian matrix. In practice this matrix is computed using the second-order finite-difference method using small perturbations of the baseflow, refer to [17] for more details and appendix 9.3. The absolute stability of steady state $\underline{q}_{BF}$ is then determined by the existence of at least one eigenvalue having a positive real part in the spectrum of this matrix. Figure 4.35 (top center) represents some of these eigenvalues in the complex plane. As $\underline{\underline{J}}$ is real-valued, its spectrum is symmetrical with respect to the horizontal axis, then for the sake of simplicity, only its "upper" part is represented. Concerning the non-controlled baseflow, one can notice the presence of four unstable eigenvalues, which is consistent with the results of the study of Sipp & Schmid [228] on the same geometry and Reynolds number but on an

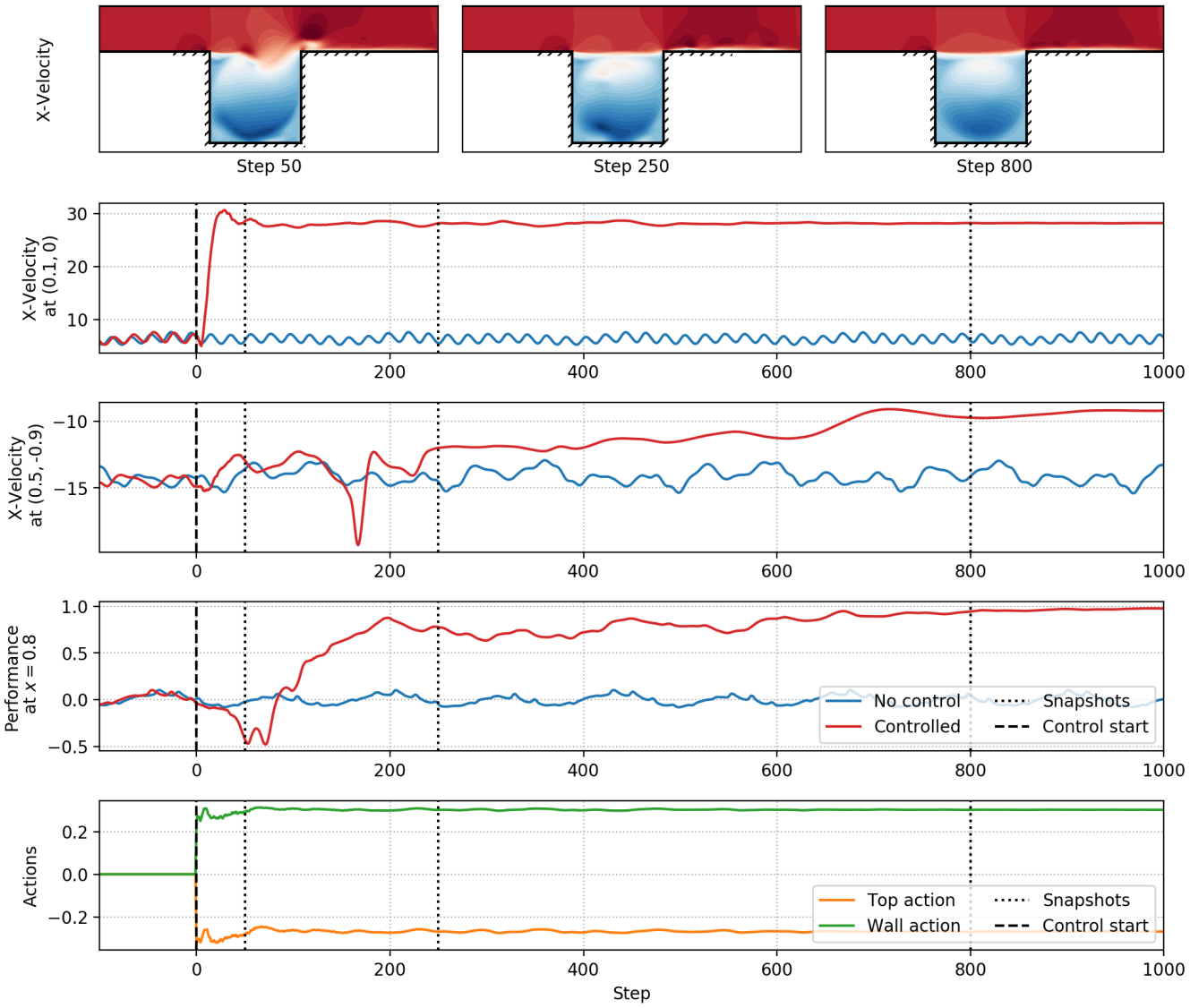


Figure 4.34: Evolution of multiple indicators during an evaluation run of the best individual of the test-batch. (top graphs) Horizontal velocity flow snapshots. The color scale is the same as the one used in previous flow figures 3.19 and 4.30. (bottom) Evolution of both control actions with time. (other graphs) Evolution of the metric of the controlled flow (red line) compared to the non-controlled baseline (blue line).

incompressible flow. The modulus of the eigenmodes corresponding to these unstable eigenvalues is represented on the graphs below. Eigenvalues of both flows can be paired, as shown on the figure, refer to appendix 4.4.3.7 for more details. Despite having differing Strouhal numbers (here non-normalized) and amplification rates σ, these modes present a very similar spatial structure, mostly peaking near the downstream edge of the cavity. The spectrum of the controlled flow is superimposed on the same graph, one can likewise identify three eigenvalues that stand out. This time, only one is unstable and one is marginally stable. The structure of the corresponding modes also peak in the same area as previously but one can still notice slight modifications of their structure due to the modification of the baseflow by the control action. The controlled flow has thus one unstable mode and may then slowly develop an unsteady behavior if the forcing is kept constant

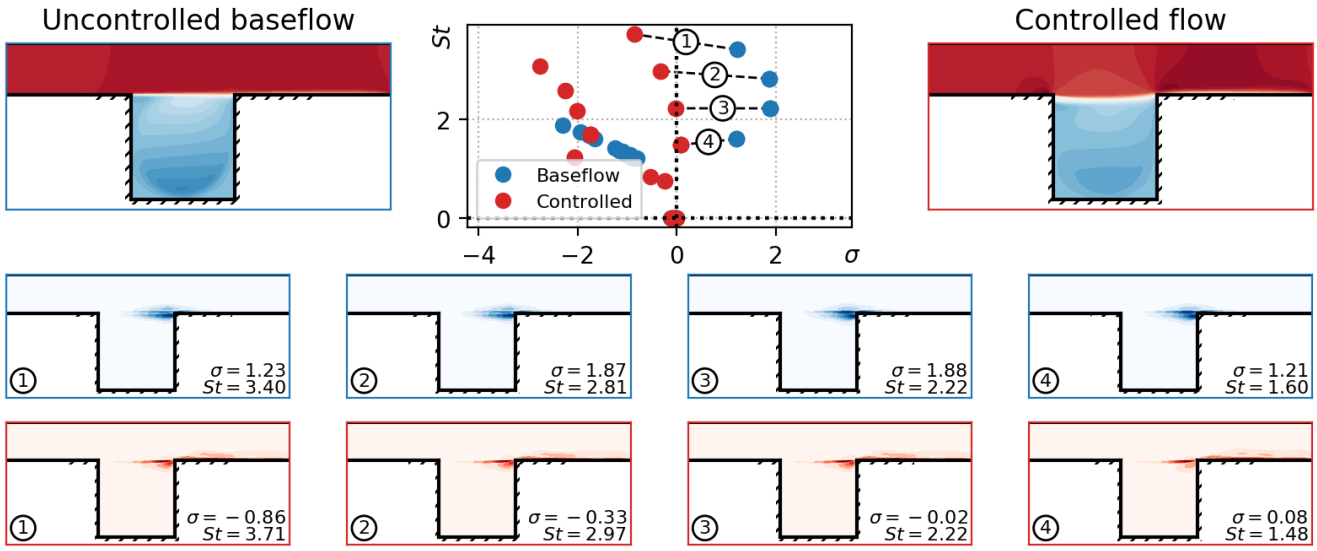


Figure 4.35: Comparison of the results from the global absolute stability of both the controlled steady flow (red-framed plot and marker) with the non-controlled baseflow (blue-framed plots and markers). (top left and right) Horizontal velocity snapshots of the non-controlled baseflow (blue-framed, left) and the steady controlled flow (red-framed, right). (top center) Eigen-spectra of the baseflow and the steady controlled flow. (bottom graphs) Modulus of the eigenmodes corresponding to the numbering of the stability spectrum for both the non-controlled baseflow and controlled flow.

(equal to its final mean value). This is later discussed in section 4.4.3.6.

4.4.3.4 Fluctuation reduction

The long-term behavior of the controlled flow is illustrated by figure 4.36. Pressure and velocity measurement signals can be compared with the non-controlled baseline. On the three sensing locations and judging on the energy content that is decayed by 6 or more orders of magnitude compared to the baseline, the stabilization is nearly perfect. The effect of the control action seems to "bend" the mixing layer downward, thanks to both the suction of boundary layer developing on the upstream wall via the top action and to the boost in horizontal momentum provided by the sustained blowing of the wall action.

The only remaining noticeable harmonics appear mainly on both sensors located in the shear layer and refer to lower frequency signals whose amplitude is way lower than what is measured on the non-controlled flow. These peaks are likely due to the remaining unsteadiness of the long-term control action of the policy.

4.4.3.5 Effect of the reward scheduling

As introduced earlier, the reference reward formulation r_t relies mostly on m_3, to which small amplitude penalties on the average pressure signal and action norm are added. This design choice can be questioned as it appears as an indirect method to achieve the control goal. The current part discusses the effects of these choices on training through a reduced ablation study. Figure 4.37 compares the learning curves of the current setup (denoted as "ref.") with two other possible choices on different indicators of the convergence. The first implies a smooth transition of the

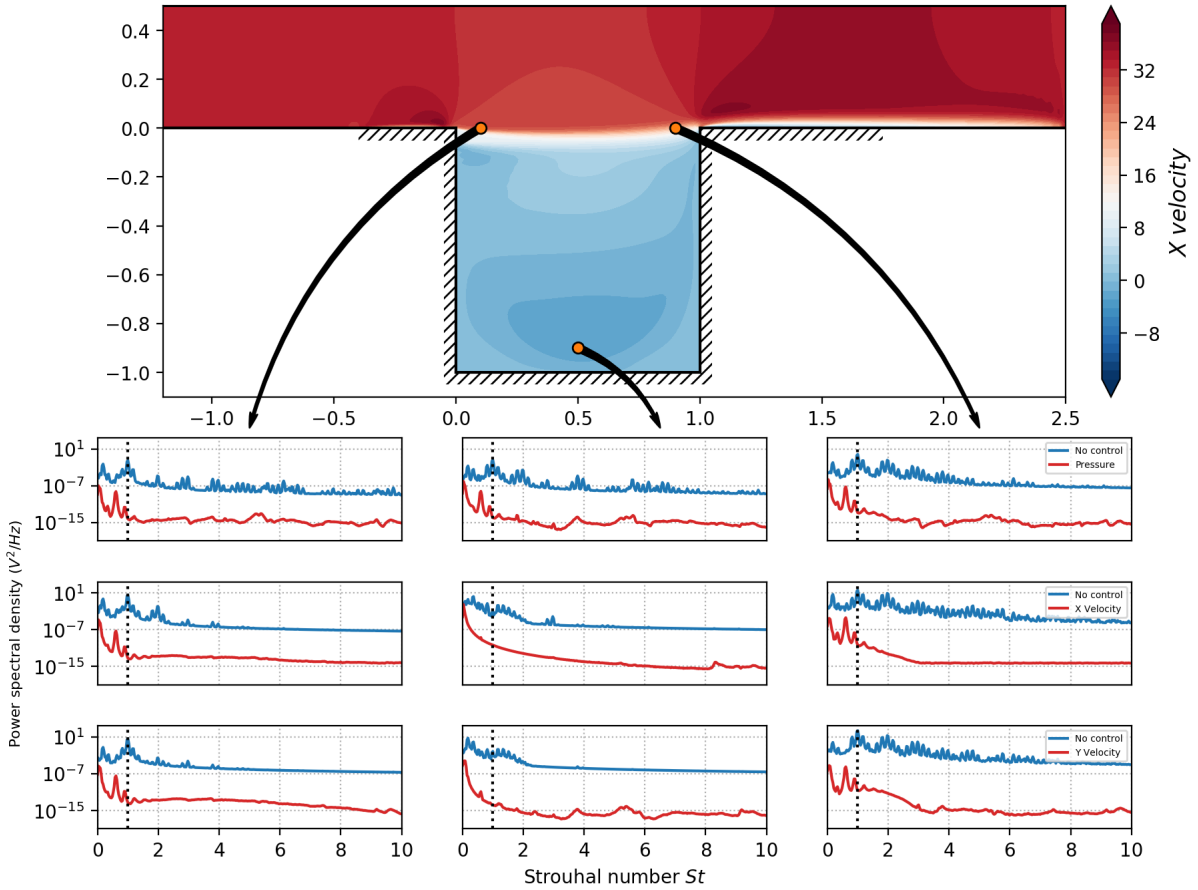


Figure 4.36: Time spectral analysis of the controlled flow. (top) Horizontal velocity snapshot of the controlled flow. (bottom graphs) Power spectral density estimation of the pressure, and velocity components of the controlled flow at three different locations (red lines), compared to the previously introduced non-controlled flow (blue lines).

reward formulation from considering m_3 to using m_8. This scheduling is performed from epoch 700 to epoch 900. The second simply considers the formulation of the reward with m_8 instead of m_3 from the beginning of the training.

First, it can be noticed that the slow scheduling setup (orange lines) performs overall worse than the reference configuration without transition from upstream to downstream fluctuation measurement. This correlates with a increase in the loss of the value function estimator, the latter quantifying the accuracy of this estimator to properly assess the expected sum of rewards in the future, i.e. the fitness of partial state s_t. It also corresponds to an increase of the average $\log \sigma$ exploration value, which end-ups matching the value observed with a direct measurement of m_8 in the reward (green lines). The latter setup shows an overall slower and poorer convergence than with the reference configuration. This reveals that measuring the reward more downstream, thus with an increased delay between forcing actions and observable consequences on the flow fitness, makes the estimation of its value harder. All the more that partial observations are much more upstream than the reward metrics. Thus, even though m_3 is an imperfect metric of the overall unsteadiness of the flow state (it leaves possible downstream-growing perturbations unmeasured), it still provides a relevant indicator of the present and future perturbations in the flow. This seems to be enough for some agents to converge toward the control objective.

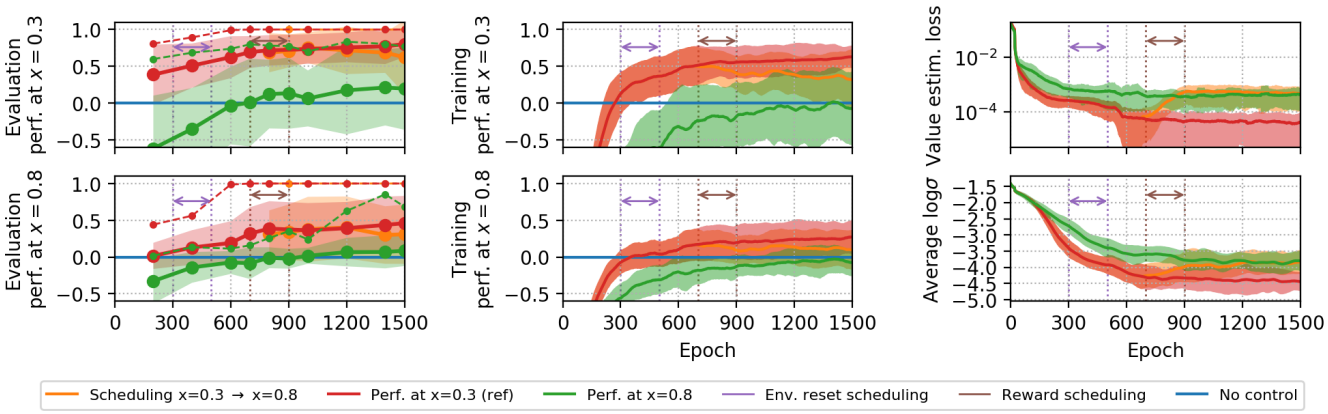


Figure 4.37: Learning curves of the last measurements of m_3 (top left and center) and m_8 (bottom left and center) metrics respectively on 4000-step evaluation runs and on training roll-outs. (top right) Evolution of the value function loss. (bottom) Evolution of the average $\log \sigma$ provided by the actor. Different scheduling setups on the reward formulation are compared. Solid lines represent ensemble averages (over 20-test-case batches) and shaded area illustrate the corresponding standard deviation.

4.4.3.6 Comparison with open-loop, and constant control laws

The study of successful control policies has shown that after a short transient, control actions are nearly steady suction (top action) and steady blowing (wall action). One can question the actual closed-loop nature of the control law and the usefulness of such computationally-intensive optimization methods if they provide trivial steady open-loop forcings.

A comparison between the closed-loop policy and other derived open-loop policies has been led to bring elements to the discussion. Figure 4.38 synthesizes the obtained results. First, the control actions obtained in closed-loop using the trained policy have been recorded on multiple evaluation runs. These control action sequences have then been implemented in an open-loop fashion starting from flow states with varying random reset states. This makes up a first "open-loop" comparison point. Second, an even simpler open-loop forcing has been tested, consisting in implementing the time-averaged action of the closed-loop policy observed after the transient. As exemplified by figure 4.38, these different forcings provide equivalent performances during the transient (up to control step 500), but fail to damp the remaining fluctuations and even allow for a slow amplification of these, leading a drop in performance. The flows obtained at the end of randomly picked runs show that, contrary to the closed-loop-controlled flow, the upstream structure of the flow is different from the non-controlled flow but unsteady perturbations could still slowly develop further downstream.

At last an hybrid control law has been evaluated, consisting of the closed-loop policy until control step 2000 then a smooth transition to the constant open-loop forcing previously introduced. This control obviously reaches the same performances as the closed-loop in the first steps. But it slowly diverges from it from step 2000 onward. Thus, even when a steady controlled flow is reached, a steady control forcing fails to prevent destabilization. This is consistent with the stability analysis of this controlled flow previously discussed.

This study then brings an answer to question of the "closed-loop" characteristic of the obtained policy. Both transient control actions and apparently-steady control actions implemented afterwards perform better than open-loop forcing. They yield slightly more efficient transients and more importantly ensure long-term flow steadiness. Thus, the RL closed-loop policy brings the flow

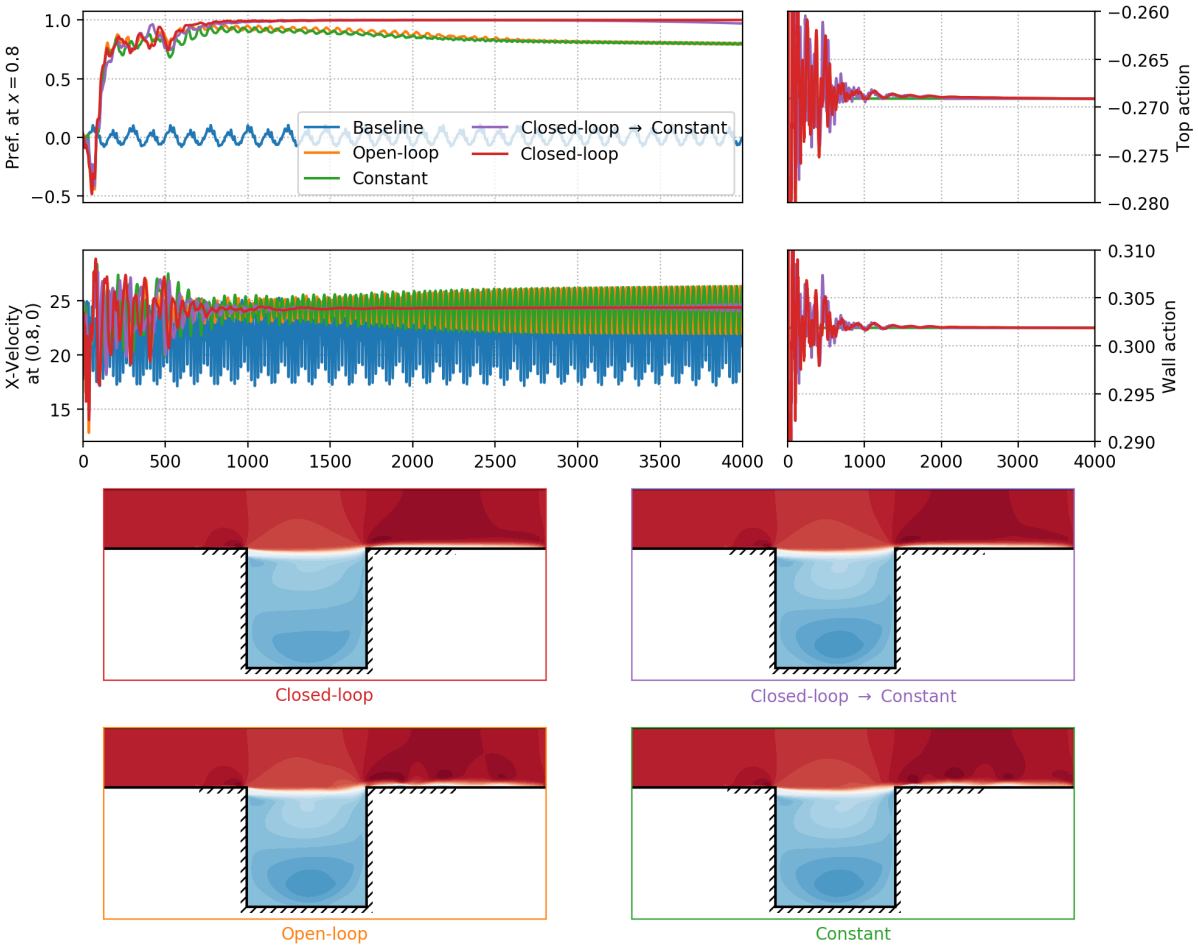


Figure 4.38: Comparison of the obtained control policy with multiple "open-loop" versions of it (described in the *ad hoc* paragraph). For the sake of readability, only one randomly-picked run is represented on the figure. (top left) Normalized performance metric m_8 and horizontal velocity signals at location $(0.8, 0)$. (top right) Forcing actions. (bottom graphs) Horizontal velocity flow snapshots obtained at the end of the test run with the different control laws.

toward a slightly unstable state, and then ensures with very slight action fluctuations that no unsteadiness appear.

4.4.3.7 Constantly-forced flow stability analysis

The long-term control action tends to stabilize the flow using nearly steady forcing actions. From the analysis of figure 4.35, unstable modes seem shifted toward stability by the control action, through a modification of the baseflow. To further study this fact, the baseflow of cases where a steady open-loop forcing, proportional to the final, time-averaged control action a_{ref} obtained using the closed-loop policy (i.e. $a_t = f \times a_{ref}$), is studied. For multiplicative factors f on this control action ranging from 0 to 2, the baseflows have been compared and their stability has been assessed. Results are reported in figure 4.39. First, one can confirm that pairs of eigenvalues really belong to the same stability branch in the complex plane.

Baseflows also appear to continuously deflect their shear layer downward with the increase in control action strength f. As shown by the hatched region on the color-bar, for actions ranging

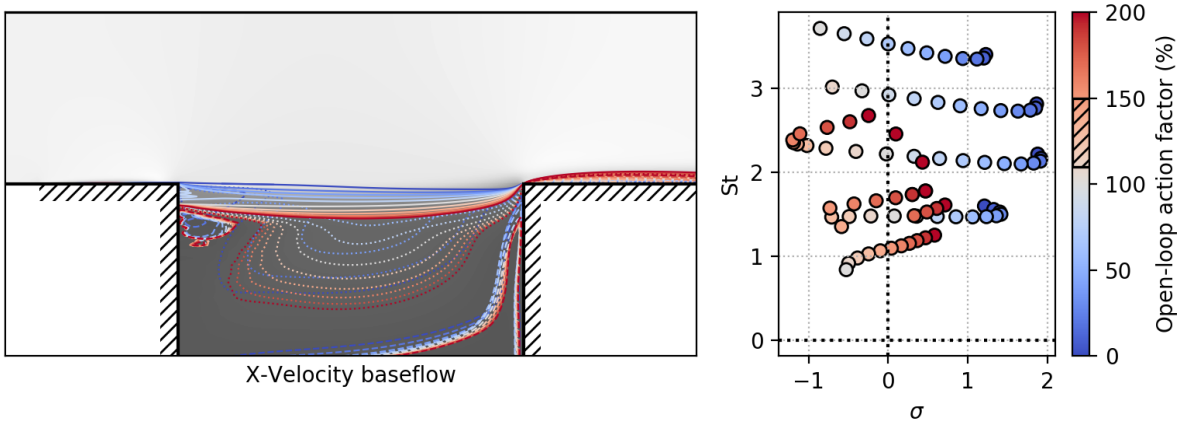


Figure 4.39: (left) Horizontal velocity iso-lines at $v_x = -0.1$ (dashed lines), 2 (dotted lines) and 8 (solid lines) plotted for different control action factors f. The correspondence between color and value is reported on the right-hand side color-bar. (right) Unstable branches of the global stability spectra for different values of open-loop forcing actions. For factors in $[110\%; 150\%]$, all eigenmodes are stable (negative real part of the corresponding eigenvalue).

from 110 to 150% of the nominal control actions, the baseflow has no unstable eigenmodes. It is thus likely that constant open-loop forcing with these control values successfully prevent the flow from destabilizing, if they manage to first damp the initial developed fluctuations. For values of f larger than 150% the baseflow becomes unstable again.

4.4.3.8 Control robustness

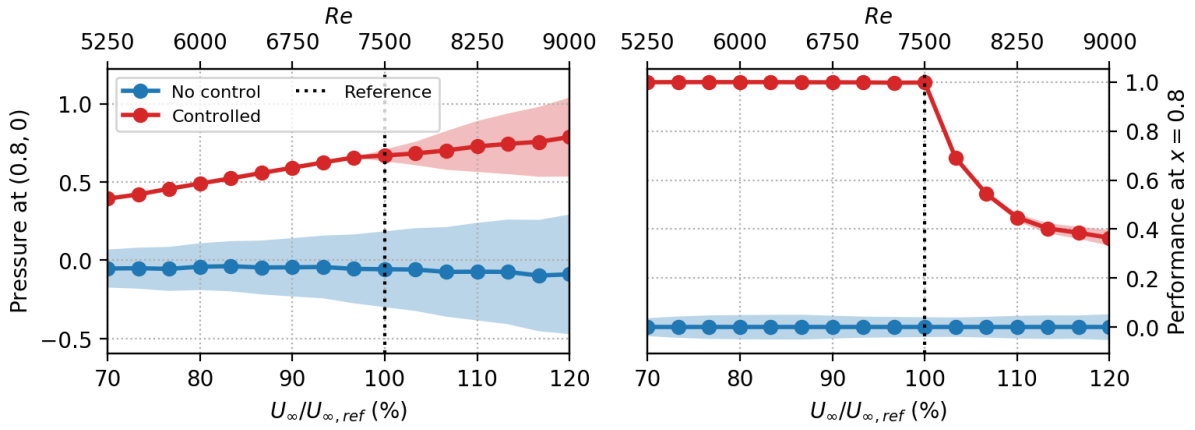


Figure 4.40: Robustness to a change in the free-stream velocity. (left) Comparison of the relative measured pressure at $(0.8, 0)$ between non-controlled and controlled flows. (right) Final value of the normalized performance metric m_8 at the end of 8000-steps evaluation runs. Solid lines represent time averages whereas shaded areas illustrate the average standard deviation observed over windows of 40 control steps.

The effect of a change in flow regime has been tested by the modification of the free-stream velocity U_∞. All things otherwise equal, this affects the Reynolds ($Re = \frac{U_\infty D}{\nu}$) and Mach numbers

($M_\infty = \frac{U_\infty}{\sqrt{\gamma r T_\infty}}$). When varying U_∞, the numerical time-step is adapted but the control step physical duration is fixed. Action and observation setups are also kept unchanged. The best RL-trained closed-loop policy (converged at $Re = 7500$) has been evaluated on cavity flows where the free-stream velocity U_∞ ranges from 70 to 120% of its nominal value. This corresponds to a range of Reynolds numbers from 5250 to 9250 and upstream Mach numbers from 0.07 to 0.12. The results of this study are presented in figure 4.40. One can notice that for velocities lower than the reference, the control policy successfully reaches the maximum stabilization performance and keeps the flow steady, where for velocities larger than U_∞, the performance degrades. Here evaluation runs last for 8000 control steps, thus the final measured performance is not statistically steady for Reynolds number larger than 7500 displaying very long destabilization transients after an initial successful stabilization of the flow (not shown).

Figure 4.41 illustrates the robustness of the control policy with respect to additional measurement noise which is implemented as follows. If s_t and $\tilde{s}_t$ denote the noise-free and noisy partial observation vector respectively, one can define the noise amplitude σ so that:

$$\tilde{s}_t = \sigma u + s_t,$$
$$\text{where } u \sim \mathcal{N}\left(0, \text{std}_{\text{no control}}\right),$$

and where $\text{std}_{\text{no control}}$ is the standard deviation of the non-controlled observation signal, i.e. the baseline fluctuation level, and the random vector u being sampled at the end of every control step. This issue of robustness to observation noise has already been discussed for the control of the low-Reynolds cylinder flow discussed in section 4.2.3. It has been demonstrated that policies trained on noise-free measurements tended to perform better than the ones trained on noisy environments, when evaluated on noisy environments. This suggests that the exploration noise is sufficient to ensure policy robustness even against much stronger measurement noise levels.

As shown by figure 4.41, control performance is rather resilient to observation noise. Up to $\sigma = 0.1$ (i.e. 10% of the non-controlled observation fluctuation levels), the performance converges toward its maximum value. For larger σ values, performances decrease but still remain better than non-controlled baselines, despite strong fluctuations of the agent's input, thanks to a "damping effect" of the actor, concerning the control action.

4.5 Synthesis

All the presented results would not have been achieved without preemptive hyper-parameter searches. Generally speaking, flow control environments are more sensitive to the hyper-parameter configuration than simpler and cheaper environments. As sample cost also represents a major difference compared to these "sand-box" environments, these fine-tuning search phases were computationally costly. Still, RL has shown its ability to provide effective, robust and energy efficient control policies on high-dimensional environments displaying non-linear dynamics.

From these findings, multiple issues concerning the application of RL and LGP algorithms arise. Sample efficiency is key to algorithm performance. It should be optimized with care since computational cost quickly becomes the main barrier to the application of these methods to more complex and costly flows.

In the same perspective, both sensor and actuator parsimony are matters of concern for two mains reasons. First it has been observed that convergence accelerates as the number of agent

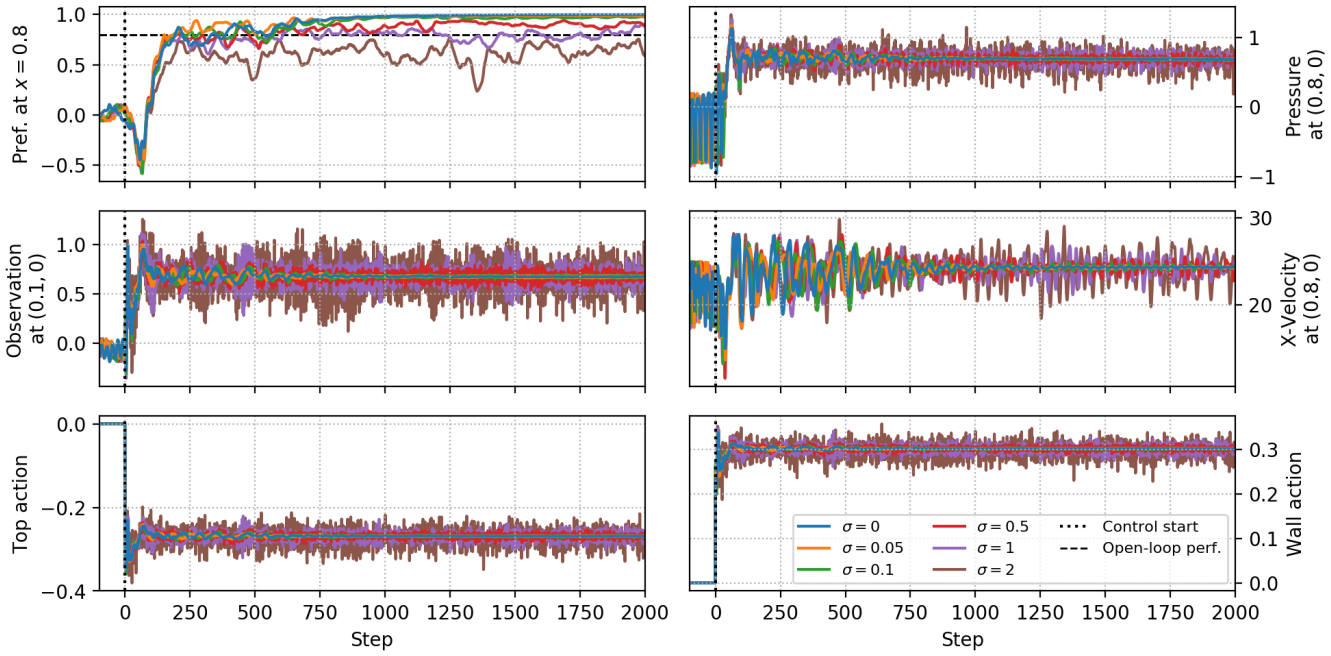


Figure 4.41: Robustness to Gaussian noise on partial observations. (top left) Evolution of the normalized performance metric m_8 for different noise levels. The average final performance of open-loop runs discussed in section 4.4.3.6 is added for comparison (black dashed line). (center left) Upstream pressure observation located in $(0.1, 0)$. (top right) Downstream pressure measurement in $(0.8, 0)$. (center right) Horizontal velocity measurement in $(0.8, 0)$. (bottom left and right) Top and wall control actions respectively.

inputs and outputs decreases. Having less sensors and actuators is thus a leverage to reduce the computational costs whose importance has been emphasized early on. The second reason concerns the transposition to experimental or industrial contexts where real-world constraints such as hardware availability or sensing and actuation limitations are important factors considering the feasibility of a given application. There is indeed a vested interest in having sparse sensors and actuators layouts, but this should not come at the cost of performance. Thus methods enabling the first while preserving the latter as much as possible would represent a significant step forward for the maturity of these control methods.

The following chapters successively tackle the sensor and actuator sparsity issues. Chapter 7, addresses transverse issues encountered on these test-cases that also impact sample-efficiency.

Chapter 5

Sensor sparsity

This section tackles the issue of sensor layout parsimony. After an introduction of the existing literature on this topic and a justification of the theoretical and practical interest of sensor sparsity, a RL-based method, proposed in Paris *et al.* [174], and it results are detailed. **Parts of this chapter are thus directly drawn from this article.**

Contents

5.1 Problem statement

5.1.1 Relevance of the issue

Reducing sensor requirements while keeping optimal control performances is crucial to potentially transpose flow control techniques to experimental and industrial cases. Contrary to numerical simulation, where the full flow state is available, experimental setups and further real-world applications of flow control see much more constrained measurement options. Pointwise probes are constrained by their physical footprint and must be integrated in a way that minimizes disturbances on the flow. This makes wall-mounted measurements more desirable over intrusive "wall-detached" ones. Second, even if they are wall-mounted, cable-routing and local wall geometry may still prevent laying sensors at a desired location. Volume or surface measurement using methods such as Schlieren imagery, Particle Image Velocimetry (PIV) may as well be constrained by their footprint. These methods may also require extensive post-processing, which may compromise real-time applications for control[1]. Overall, flushed sensors (such as pressure or temperature probes) or wall-mounted ones (such as Pitot probes) may be among the only realistic measurements one can hope for in the context of an industrial application of closed-loop flow control.

Aside from location and measurement techniques, the number of sensors may also be an obstacle to scaled-up applications. From the practical point of view, more sensors mean more hardware (the sensors themselves but also the whole acquisition and post-processing pipeline). Reducing the number of these, may enable smoother transitions from numerical setups to experiments and prototypes as well as reduced costs. Second, from a more theoretical perspective, fewer measurements make-up a reduced number of inputs to control laws and agents. This generally makes training faster and reduces failure modes due to unexplored regions of the state space since the latter see its dimension reduced. One can expect slightly less efficient but more robust control laws when reducing the sensor footprint.

For instance, Rabault *et al.* [185] and Tang *et al.* [237] respectively use 151 and 236 probes to control the flow past a 2D-cylinder. Their work is therefore a first step for DRL control that needs to be continued and further improved, especially concerning this issue of sensor layout. There is interest into having methods that enable optimized sensor layouts while preserving performance as much as possible.

5.1.2 Existing literature

This issue of optimal measurement location has been investigated by many authors outside of the context of DRL and mainly in a linear framework, for instance by Mons *et al.* [160, 159], Mons & Marquet [161], Foures *et al.* [72], Verma *et al.* [241] for data assimilation. Bright *et al.* [32] took advantage of compressed sensing to perform flow reconstruction using a limited number of sensors. The optimal estimation of a reduced order state, usually POD modes, has been used by Cohen *et al.* [54], Seidel *et al.* [214] and echoes the assumption that accurate flow estimation is an essential feature of efficient control. However, as stressed by Oehler & Illingworth [170], control does not systematically require faithful flow reconstruction (in the sense of POD), the partial knowledge of relevant "hidden" variables may be sufficient. This idea is conveyed (in a linear framework) by the notion of observability Gramian, introduced in section 2.1.5.2. Empirical observability Gramians

[1]Infrared thermography may be an interesting means, since it requires only moderate post-processing, but the geometry should allow for appropriate camera angles

were used by Singh & Hahn [223], DeVries & Paley [64] for flow estimation and Manohar *et al.* [150] leveraged balanced POD to design an optimal H_2 control of a linearised Ginzburg-Landau model.

In their multi-step heuristic approach to closed-loop flow control design, Seidel *et al.* [213] propose to rely on the correlation between observed relevant phenomena and sensor signal to choose observations and on clues provided by POD mode decomposition of the flow instabilities to choose actuators location. Cohen *et al.* [54] and Willcox [249] also leverage POD analyses to derive optimized sensor placement. Multiple studies rely on adjoint sensitivity analysis [52] and on the "wavemaker", the overlap between the direct and adjoint sensitivity modes, introduced by Giannetti & Luchini [78] to derive appropriate sensor (and actuator) placement. Li & Zhang [140] used a computation of the wavemaker on a confined cylinder flow to lay their sensors used as feedback for a reinforcement-learning-trained policy. Sashittal & Bodony [206] applied a related method on a data-driven, linearized model of their systems to position their sensors. They applied this method to control both the linearised complex Ginzburg-Landau equation and the flow over an inclined flat plate.

5.2 The proposed approach

Based on the paradigm of RL, deemed as an appropriate framework for searching optimized sensor layout, the approach proposed here, leverages sparsification. Using a pre-trained agent on a "fully developed" measurement layout, the algorithm seeks to cut-off a prescribed number of inputs while keeping optimal control performance. In the following the additional neural structure tasked with sparsification is introduced, then its training method is detailed before a discussion of the results of the proposed method.

5.2.1 Sparse surrogate actor

The proposed method (called Sparse-PPO-CMA, or S-PPO-CMA) leverages policy distillation and splits into two separate phases: training a conventional PPO-CMA actor-critic structure (described in part 2.2.5), then deriving a sparse surrogate actor that imitates the actions of the original actor with fewer input signals. The pre-trained agent's structures are denoted by π^* for the actor and V^* for the critic. As described by figure 5.1, the sparse actor π_s is composed of a dense neural network having the same structure (architecture and activation functions) as π^*, to which a Stochastic Gating Layer is added.

During the sparse training phase, the action a_t is either sampled using the optimal policy π^* or π_s, using a Bernoulli random variable. Both training of π^* and V^* are stopped, but their outputs are used to train π_s and the SGL that make up the sparse version of π^*.

5.2.2 The stochastic gated layer

The SGL mechanism used here is inspired by the stochastic gate model proposed by Louizos *et al.* [145]. Let n be the number of sensors (or the dimension of the observation space). The SGL, presented in figure 5.2 is a special simply connected layer that provides inputs to π_s and that contains substitute values $\underline{\bar{s}} = (\bar{s}_1, \bar{s}_2, ..., \bar{s}_n)$ for each observation component. Every time an observation vector $\underline{s} = (s_1, s_2, ..., s_n)$ is received, the SGL samples a random vector $\underline{p} \in [0; 1]^n$ that determines its output $\underline{\tilde{s}}$ such as:

$$\underline{\tilde{s}} = \underline{p} \odot \underline{s} + (1 - \underline{p}) \odot \underline{\bar{s}}, \tag{5.1}$$

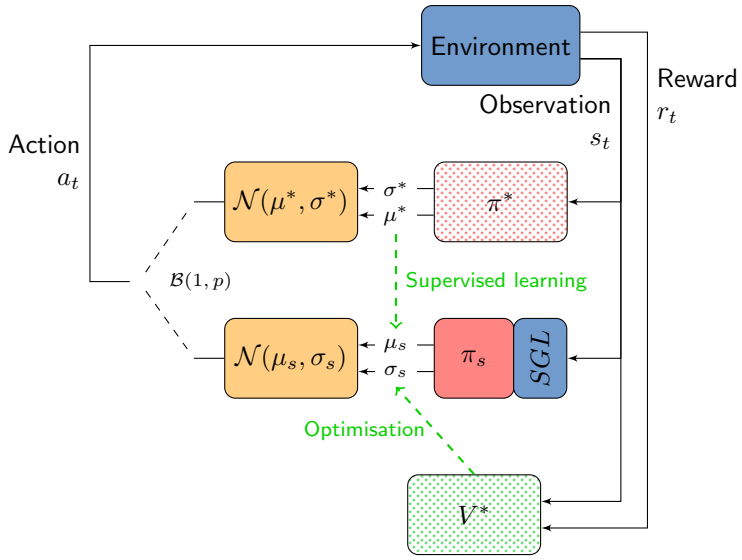


Figure 5.1: Sparse Proximal Policy Optimization with Covariance Matrix Adaptation (S-PPO-CMA). Actions are either sampled using the reference actor π^* or the sparse actor π_s via a Bernoulli choice $\mathcal{B}(1, p)$. π_s is updated via learning on σ_s using values from V^* and by supervised learning on μ_s using μ^* values. The parameters of the Stochastic Gating Layer are also updated during this phase.

where $\odot$ represents the element-wise product. Thus, $p_i = 1$ outputs the observation s_i whereas $p_i = 0$ gives its substitute value $\bar{s}_i$, and any value in-between provides a linear combination of s_i and $\bar{s}_i$. Similarly to Louizos *et al.* [145], $\underline{p}$ is sampled over a "gating" distribution f:

$$\underline{u} \sim \mathcal{U}^n(0, 1), \tag{5.2}$$

$$\underline{p} = f(\underline{u}, \underline{\alpha}) = \text{clip}\left((\zeta - \gamma)\text{Sigmoid}\left[\frac{1}{\beta}(\log \underline{u} - \log(1 - \underline{u}) + \underline{\alpha})\right] + \gamma, 0, 1\right), \tag{5.3}$$

with $\mathcal{U}^n(0, 1)$ denoting a uniform distribution on $[0, 1]^n$, β, γ, ζ being fixed numerical parameters, $\underline{\alpha}$ being a trainable vector steering the expectation on $\underline{p}$ and $\text{clip}(a, b, c) = \min(\max(a, b), c)$. f can be seen as a "soft" Bernoulli choice distribution enabling values of $\underline{p}$ in $[0; 1]^n$. The L_0 complexity of the SGL, giving the expected number of observation components s_i for which $p_i > 0$, can be written as:

$$\mathcal{L}_c(\underline{\alpha}) = \sum_{i=1}^{n} P(p_i > 0) = \sum_{i=1}^{n} \text{Sigmoid}\left[\alpha_i - \beta \log \frac{-\gamma}{\zeta}\right]. \tag{5.4}$$

During testing, $\underline{p}^*$, the most likely value of $\underline{p}$ is chosen deterministically as:

$$\underline{p}^* = \text{clip}\left((\zeta - \gamma)\text{Sigmoid}\left[\underline{\alpha}\right] + \gamma, 0, 1\right). \tag{5.5}$$

Both $\underline{p}$ and $\underline{p}^*$ can take values between 0 and 1 (included), thus modeling a fully "open" or fully "closed" gate while still allowing for a gradient-based optimization using the loss $\mathcal{L}_c$.

5.2.3 Loss formulation and sparse actor training

The weights θ_s of the sparse actor π_s are initialized using the weights θ^* of π^* and updated every epoch both by the training of σ_s and μ_s. σ_s is trained in the same way σ^* has been trained (refer

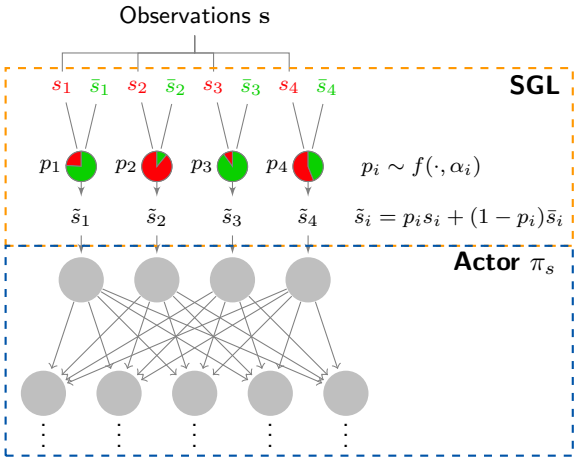


Figure 5.2: Stochastic Gated input Layer (SGL). Received observation s_i is either passed on to the actor π_s if $p_i = 1$, combined with its substitute value $\bar{s}_i$ if $p_i \in]0; 1[$, or replaced by $\bar{s}_i$ if $p_i = 0$. p_i is sampled using a "gating" function f parameterised by α_i, which is updated during the second phase of the S-PPO-CMA method.

to section 2.2.5). Concerning μ_s however, a supervised learning using the optimal action μ^* is performed with the loss:

$$\mathcal{L}_{\pi_s}(\theta_s, \underline{\alpha}) = ||\mu_s(\underline{s}, \theta_s, \underline{\alpha}) - \mu^*(\underline{s}, \theta^*)||_1. \tag{5.6}$$

For the SGL, $\underline{\alpha}$ and $\underline{\bar{s}}$ are trained in the same process as θ_s, allowing π_s to "adapt" to the variations of input $\underline{\tilde{s}}$ caused by the updates of the SGL. $\underline{\bar{s}}$ is slowly updated using the observation values $\underline{s}$ at every epoch and updates of $\underline{\alpha}$ are based on the following loss $\mathcal{L}_{\text{sparse}}$:

$$\mathcal{L}_{\text{sparse}} = \mathcal{L}_{\pi_s}(\theta_s, \underline{\alpha}) + \lambda \left[\mathcal{H}_1(\mathcal{L}_c(\underline{\alpha})) + \underline{\underline{\Gamma}}\underline{\alpha}\right], \tag{5.7}$$

where λ is the regularization parameter, $\mathcal{H}_1$ is a unitary Huber loss and $\underline{\underline{\Gamma}}$ can be seen as a Tikhonov matrix that accounts for strong correlations between observations. Its purpose is to penalize α_i whose observation s_i is correlated with any other $s_{j \neq i}$ and thus is redundant (refer to Paris *et al.* [174] for more details). The choice of λ drives the equilibrium between sparsity and control performance.

5.3 Results on the cylinder flow

5.3.1 A first ablation study

In this part, a systematic study on sensor configurations within a 3-by-5 grid-like layout ($(x, y) \in \{1, 2, 3, 4, 5\} \times \{-0.5, 0, 0.5\}$) is performed. Figure 5.3 illustrates the learning curves of 10-case batches having from 3 to 15 sensors. The addition of the second and third columns of sensors (located in $x = 2$ and $x = 3$) yields a significant gain in performance, and one can notice that 12 and 15-sensor layouts have a very similar average performance. Thus it is possible to conclude that the three additional sensors (located in $x = 5$) are not useful to the control strategy.

Figure 5.4 shows the effect of the location of pressure observations, for an array of 6 sensors that are displaced in the streamwise direction. This time, the importance of the first sensor column (located in $x = 1$) is demonstrated by the noticeable gain in drag reduction between the first two layouts (blue and yellow curves). The importance of the third sensor column ($x = 3$) is once again

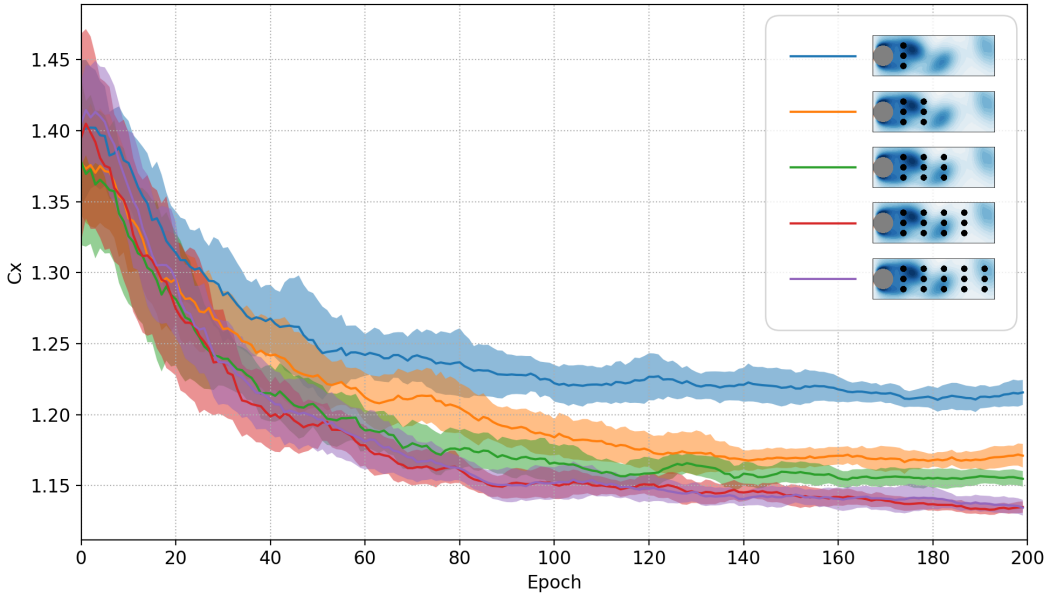


Figure 5.3: 10-case batch-averaged learning curves for different sensor layouts. Shaded areas represent the standard deviation of the corresponding plotted quantities. Each C_x value is averaged over the whole epoch, including the transient from developed vortex shedding to controlled flow. This explains the discrepancy with pure performance values on C_x earlier introduced.

stressed by the decrease in performance between the green and red curves. Within this predefined combinatorial set, this partial study highlights the relevance of sensors closest to the cylinder. These first preliminary tendencies are confronted in the next section with the results from the newly-proposed S-PPO-CMA algorithm that is designed to provide an optimized sensor location for the control.

5.3.2 Optimized choice of sensors

The S-PPO-CMA algorithm, is used here to derive optimized sensor placement for any allocated number of sensors ranging from 1 to 9, within the imposed 15-sensor grid-like pattern. The number of sensors at convergence is indirectly controlled through the value of the L_0 regularization constant λ, that balances the gradients of both $\mathcal{L}_{\pi_s}$ (performance loss) and $\mathcal{L}_c$ (complexity loss). Figure 5.5 shows the achievable drag reduction with respect to the number of sensors i and the corresponding sensor layout l_i. Note that due to the symmetry of the configuration, there always exist pairs of symmetric layouts that achieve identical performances. The S-PPO-CMA algorithm randomly outputs one of the two optimized layouts for each value of λ, but only one layout is displayed in the figures for simplicity.

With a single sensor, the drag reduction is around 11% and it peaks to approximately 18% for 5 sensors or more. The sensor pattern's tendency to fill without relocating existing sensors, meaning that $l_i \subset l_{j>i}$, is a sign of convexity of this problem in the sense that any combination of the optimized layout set is also part of this set. It is interesting to notice that, starting from the 5-sensor optimized layout, the addition of more sensors does not improve drag reduction, which makes this 5-sensor layout the optimized trade-off between performance and sensor setup complexity. This

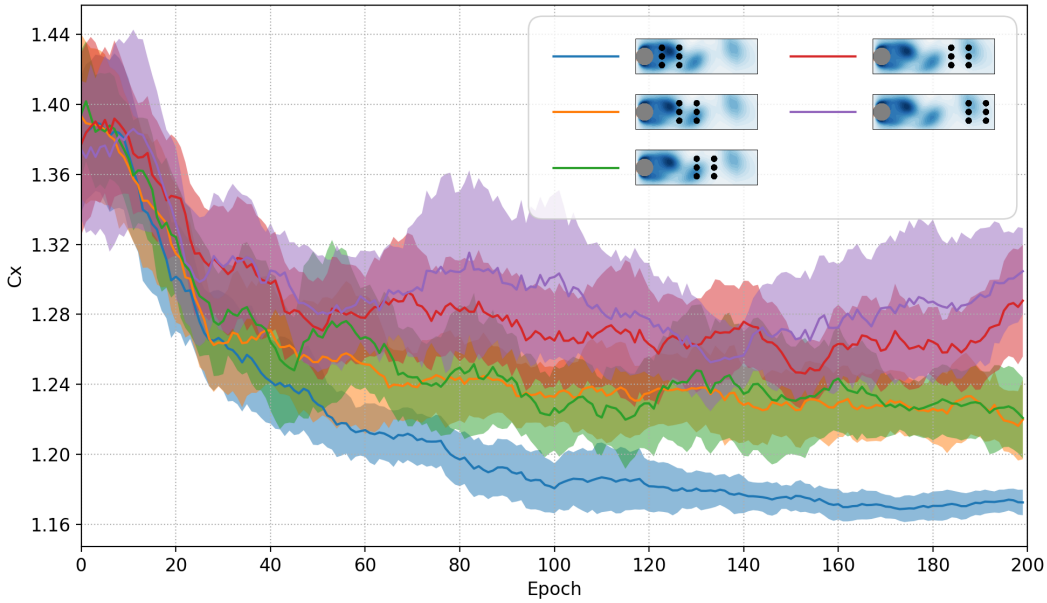


Figure 5.4: 10-case batch-averaged learning curves for different sensor layouts. Shaded areas represent the standard deviation of the corresponding plotted quantities. Each C_x value is averaged over the whole epoch, including the transient from developed vortex shedding to controlled flow. This explains the discrepancy with pure performance values on C_x earlier introduced.

layout is not reminiscent of anything used in the large number of existing studies on the control of the 2D cylinder wake. Thus, this highlights the usefulness of the S-PPO-CMA algorithm since optimized sensor placement is, even in such a simple case, not particularly intuitive.

The centerline locations ($y = 0$) do not appear relevant for the control since the corresponding sensors are never selected by the algorithm. A possible explanation may be that these sensors cannot provide information relative of the instantaneous asymmetry of flow, and are thus unfit to choose the action's sign. The first two layouts l_1 and l_2 validate the importance of the first sensor column, and the selection of sensors shows a weaker importance of locations beyond $x = 4$. This is in line with the conclusions of section 5.3.

As discussed previously, many studies optimize sensor placement based on the linear framework of POD, with the underlying idea that the better the estimation of mode coefficients is, the better the reconstruction and control performance are. They naturally often choose locations where the POD modes are strong. Figure 5.6 illustrates the superimposition of the sensor locations with the first three POD modes derived from the natural transient from base flow to fully developed vortex shedding. These three modes account for more than 95% of the transient's energy. Despite that these modes are only valid for control trajectories that stay close to this natural transient, the choice of l_2 seems reasonable as it allows estimations of both shift mode and second vortex shedding mode simultaneously, since sensors are close to the extrema of this modes (refer to left and right panels of figure 5.6). The second column of sensors appears less able to provide relevant information on the shift mode. Figure 5.6 also confirms that the centerline sensors are unfit to estimate von Kármán modes, which account for the instantaneous asymmetry of the vortex shedding.

Comparing the second and third layouts l_2 and l_3, it appears that, given the first two probe locations, an additional sensor is preferred in the third column rather than in the second. This

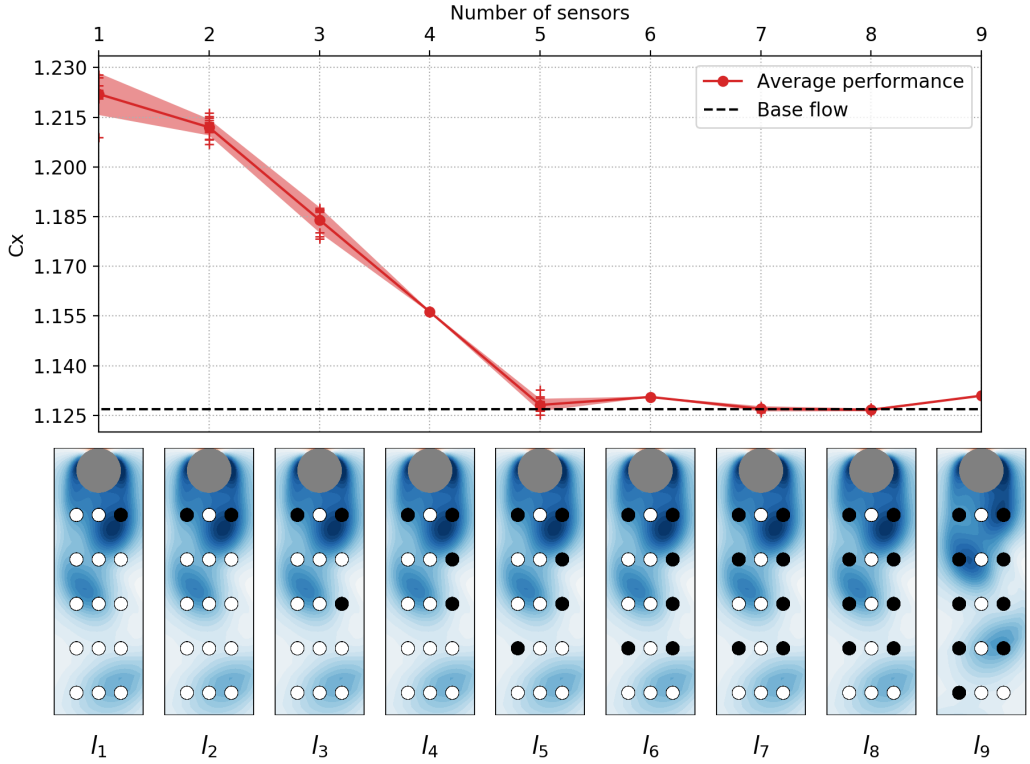


Figure 5.5: Evolution of both drag reduction and optimized sensor layout with the number of sensors. Layout thumbnails l_1 to l_9 are rotated 90° clockwise.

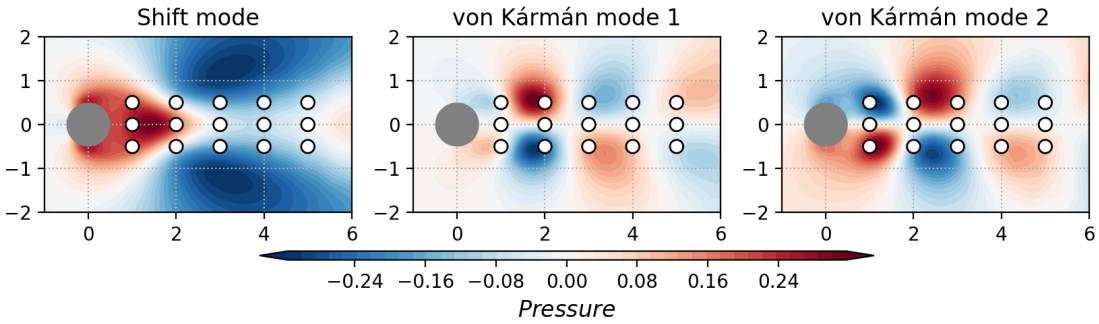


Figure 5.6: Comparison of the sensor locations with the first three POD modes of a natural transient from base flow to fully developed vortex shedding.

might be because this layout provides a better "coverage" of the instantaneous recirculation bubble. Additionally, despite the lack of mean flow symmetry during the transient phase of control, sensors tend to concentrate on a single streamwise row. From a POD-based control viewpoint, this may not be optimal for modes reconstruction. This shows a strength of the present approach: an estimation of the full flow field (or the dominant POD modes) is likely not to be needed for the control. Therefore, searching for points that allow such a full reconstruction may be sub-optimal. In that context, favoring a precise vortex tracking, whose center travels in the vicinity of the external sensor

rows over a more complete estimation of the wake may lead to better performance.

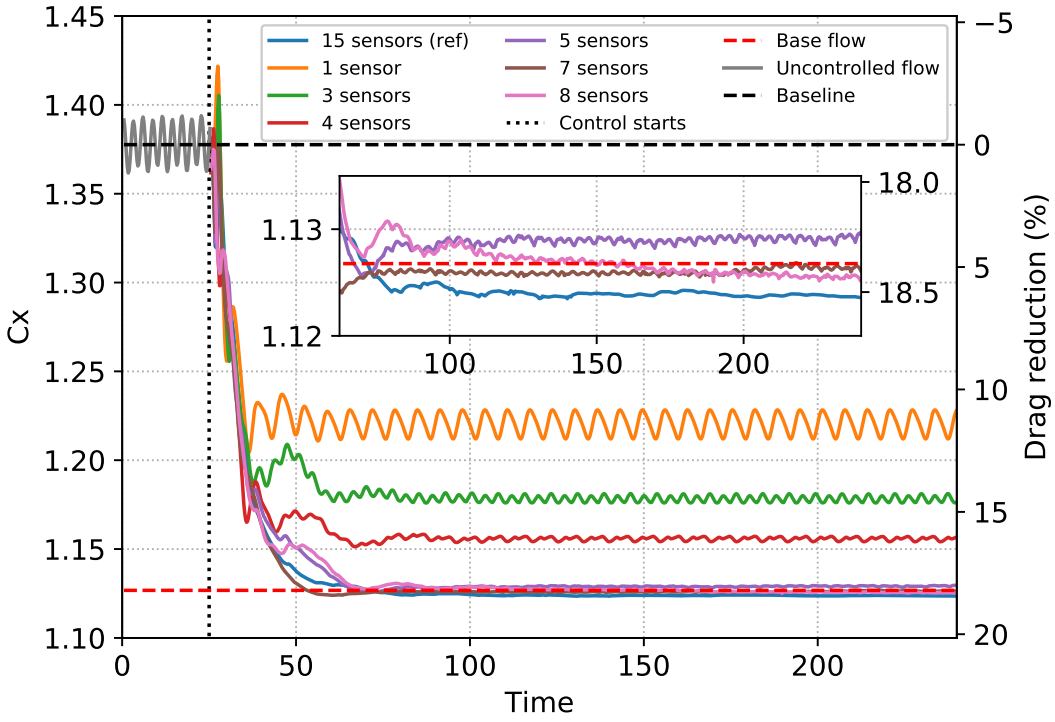


Figure 5.7: Performance comparison for different optimized sensor layouts.

Figure 5.7 compares the performances of some of the previously found sensor layouts throughout time. It confirms that as from 5 sensors, optimal performance ($\sim 18\%$) is reached. All configurations show a comparable transient phase that stops earlier both in time and drag reduction for the sparsest sensor configurations. Despite an already significant drag reduction, layout l_1 seems unable to notably stabilize the vortex shedding instability. A much better steadiness is achieved with 3, 4 and more sensors.

5.4 Synthesis

The issue of optimization of the number of sensors while preserving performances has been investigated in this chapter. The lack of sensor sparsity strategies in the context of DRL where extensive sensor layouts tend to be the norm, has led to proposing S-PPO-CMA that selects the most relevant sensors and discards redundant and irrelevant ones, using a pre-trained control agent. Tested on a 2D-cylinder flow and compared against a coarse systematic study, our proposed algorithm managed to reduce the sensor layout to only 5 individuals while keeping state-of-the-art performance. The obtained sensor layout has been compared with both the outcomes of our systematic study and conclusions of other linear (mostly POD-based) studies. Multiple explanations have been proposed to back the observed consistency of these results. This optimization is yet based on a coarse grid of sensors. A denser layout, covering a larger portion of the flow has been studied (not shown) and gives similar results. The reproducibility of this method, should later be asserted by performing

trainings on larger batches of test-cases and confronting the results against random layout samplings or systematic studies.

Chapter 6

Actuator sparsity

With the same motivation as for the previous chapter, i.e. reducing hardware needs and increasing robustness while still preserving efficient control laws, the current section discusses the issue of actuator sparsity. A first literature review introduces the current state-of-art and the specificity of actuator sparsity compared to sensor sparsity. Then a generic sequential action sparsification method is introduced. Later, three candidate implementations of this generic method, using different metrics are discussed, before turning to model-based control, that proposes a paradigm shift that may be well-suited in that case.

Contents

6.1 Problem statement

Reduced actuator footprints should, for the same reasons as for sensor sparsity, be preferred over more extensive ones. Here the question of control action feasibility is voluntarily let aside (it is obvious that volume forcing control actions are unrealistic in most cases), more details concerning actuators can be found in part 2.1.3. From the experimental perspective, hardware requirements and power consumption scaling with the number of control actions pledge for parsimony in control actions. Theoretically, a reduced number of outputs translates to a easier fundamental understanding of the control dynamics and to a reduced number of potential failing modes. Again, reducing the

state-action search-space may lead to sub-optimal control laws compared to the ones that can be reached on a more controllable setup. The critical issue here is thus to locate or select the reduced number of actuators wisely.

6.1.1 Existing literature

The vast majority of the body of work concerning actuator layout optimization relies on controllability and balanced model reduction methods. These model reduction methods were used by Bhattacharjee *et al.* [25] who take advantage of the eigensystem realisation algorithm (ERA) to compare the controllability (in a $\mathcal{H}_2$ framework) of multiple jet actuators laid on the suction of an airfoil to select the best one depending on the performance criterion (lift or angle of attack upon flow separation). Linear-quadratic-Gaussian control on balanced truncated reduced models is employed to derive optimal sensor and actuator placement using gradient descent methods [150, 261].

Oehler & Illingworth [170] and Jin *et al.* [108] used both optimal estimation and full-state information control to derive optimal sensor and actuator placement for the complex linearized Ginzburg-Landau system and a low Reynolds number cylinder wake respectively. Yeh & Taira [262] also employed resolvent-based analyses to discover the optimal forcing variable and location for an actuator aiming at preventing airfoil flow separation. Natarajan *et al.* [166] resorted to the wavemaker [78] to locate both sensors and actuators optimally in a diffuser flow.

Apart from studies in a linear framework, there has not been much work on this topic. Among non-linear actuator methods, one may cite the study of Rogers [192] who derived a set of actuator layouts on a stealth bomber to satisfy maneuverability goals using a genetic algorithm.

6.1.2 A sequential elimination paradigm

Finding the optimized actuator layout, at a given energy budget or a fixed number of actuators appears as a supplementary level of optimization, built on-top of standard RL algorithms. To this extend this task belongs to the family of meta-RL, whose methods aim at optimizing performance by modifying either the state-space, the action-space and/or the reward formulation.

Here the exploration is constrained by a kind of discretization of the actuator search space. Let us assume that the space of all possible locations of actuators is a quantified "layout space", like an array of n spots in the environment that can be either vacant or occupied by an actuator (whose action is later optimized in coordination with all the others). Thus here, for a given budget of m actuators, finding the optimal layout is an optimization problem of combinatorial complexity.

As any actuator can obviously take "neutral" control actions if these are deemed optimal and thus mimic any "child" layout that contains at most the same actuators, based on performance, one can formulate a total order relation between layouts that guarantees performance increase when adding actuators. Said more mathematically, if l_i is a set of actuators with a optimal performance P^{l_i} then:

$$\forall\, j,\ l_i \subseteq l_j \implies P^{l_i} \leq P^{l_j}. \tag{6.1}$$

This equation states that a layout containing all the possible actuators has an optimal performance that makes a relevant upper-bound of all the other layout performances. This fact helps justifying the current choice of paradigm for the exploration of this "layout space" among the choices of methods listed in table 6.1.

The first obvious exploration option is a brute-force strategy that systematically tests for all the layouts for finding the best one (with absolute certainty). This approach may be possible for cheap

Method	Type	Cost	Accuracy
Brute force	Global	+++	exact
Random sampling	Global	++	+
Sequential elimination	Local	+	+++
Elimination/activation walk	Local	++	++

Table 6.1: Qualitative synthesis of exploration strategies for actuator layout optimization.

environments such as the ones used as test bench for most of the RL studies, and with a reduced layout space, but becomes very quickly out of reach as either the number of possible actuators or the sampling costs grow. Random sampling is a less costly option. It can be accompanied by informed sampling methods such as genetic algorithms or gradient-estimation-based sampling. The latter only works in cases where linearity can be assumed between actuator activation and optimal layout performance. Generally speaking, these global, random exploration methods suffer from the "curse of dimensionality", their exploration efficiency collapsing with the dimension of the search space, all the more that these do not exploit the specific structure of the search space in the present case.

Local searches, that consist in starting from a given layout and gradually modifying it by informed choices to improve performance, appear more relevant for the problem at hand. Two aspects need to be specified for each of these methods: the starting point and the information in which updates are decided.

Among all possible starting layouts, the one with all the actuators being active stands out thanks to its overall best performance. Let us justify why this choice is relevant. If the initial layout is chosen otherwise, two cases arise. This layout may have the same number of elements (or size or cardinality) as the target. Thus exploration inevitably requires at least one activation and one elimination. These may be paired together but overall, it seems difficult to take advantage of property 6.1 from a practical point of view. If the initial layout has more (respectively less) elements than the target, the exploration can be constrained by the use of eliminations (respectively activations) only, thereby providing a clear stopping condition and a predictable convergence of the method. In the case where eliminations are performed, one can take advantage of property 6.1 to choose among the actuators. But the obtained layout is bound to be a "child" of the initial one, meaning that the initial layout strongly constrains the solution.

Using the fully developed layout, having the maximal cardinality reduces this constraint on the optimized layout to the minimum. This choice yet does not prevent from falling into local optima. In addition, and contrary to starting from a smaller layout, the convergence of intermediate policies for each layout may benefit the optimality of the starting point. And as the goal is to maximize performance, it seems reasonable to hypothesize that the best policy running on the best layout with size $n-1$ is close (in terms of performance and strategy) to the best policy running on the current layout of size n, and thus the latter is a good starting point for converging the first.

Thus starting from the overall best layout and "walking on the ridge" by elimination in the direction having the smallest slope appears as a relevant option to reach an acceptable optimum while deterministically controlling the computational overhead.

6.1.3 A harder problem than for sensors

The proposed actuator sparsity methods proposed in the current study, make use of a sequential elimination paradigm. Sensor sparsity methods rely on the idea that, sensor inputs that can be substituted by their average signal without losing to much in performance, are the ones to get rid of. Thus the sensor elimination mechanism proposed earlier runs *on-policy*, directly using the collected data. As long as full-sensor-layout policy and its sparsified version perform similar actions, they explore the same regions of the state space S. Trajectories from any of these policies are then relevant to optimize sparsity, hence the *on-policy paradigm*.

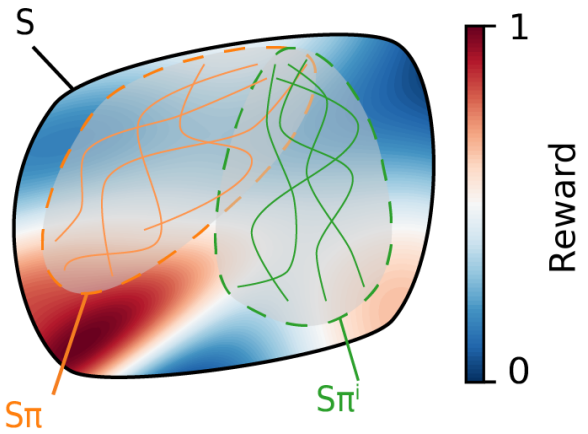


Figure 6.1: Illustrative representation of the effect of running a standard policy π, exploring the region S_π of the state space, compared to the region S_π^i explored using clipped policy π^i. Collected reward distributions are affected by clipping, and thus, so are value functions, advantages and all other stochastic estimators used by RL.

The matter is more complicated concerning action sparsity. First, one cannot build a gradient-oriented optimization of a clipping layer using a loss defined on the policy's outputs since these are bound to be clipped. Second, and as one could do concerning sensor sparsity, where simulating the effect of clipping can be done just by "pretending" not to observe (i.e. cutting off) any given set of input signals[1], one cannot "pretend" not to take control actions since these yield effects on the state transition and thus affect the distribution of the collected data. As illustrated by figure 6.1, clipping action components may lead state-space trajectories (corresponding to episodes) into different regions of the state-space S.

Thus *on-policy* data is to be considered with care since it may lead to biased estimations concerning the elimination of actuators. Hence, the need of extra *off-policy* data collection that inevitably translates into computational overhead. One is then confronted to a trade-off between cheap and possibly biased *on-policy* estimation of the impact of clipping, and different degrees of simulation of the clipping directly on the environment, that are most likely more accurate but represent extra computations. In other words, one can either **extrapolate** cheaply on the available data or **explore** the state space regions corresponding to clipping.

In this section we aim at proposing different solutions that each balance both extrapolation and exploration differently while still trying to provide optimized solutions with the lowest computational overhead possible.

[1]Observation is supposed non-intrusive, thus observing the flow or not has not impact on its dynamics

6.2 A generic algorithm

6.2.1 Process

As discussed in the previous part, the generic method proposed in this study relies on starting from a fully developed actuator layout and on the *one-by-one* elimination of actuators until the prescribed number of action components is reached. This procedure not only provides the expected layout but also all the intermediate layouts having more actuators.

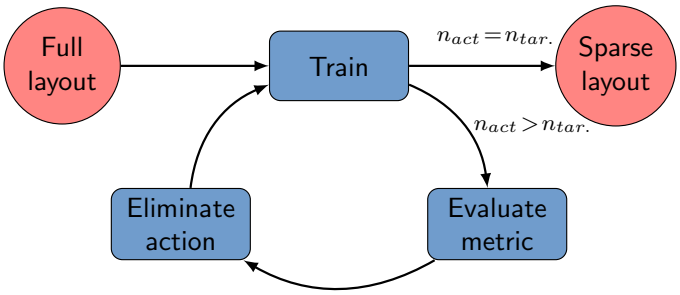


Figure 6.2: Generic actuator elimination procedure

As shown by figure 6.2, the method loops over three distinct phases:

1. A first standard training phase where the current policy is trained on the current layout to maximize its performance.

2. A second phase where *on/off-policy* data samples are collected in order to evaluate a metric used to choose the action component to eliminate.

3. A third phase where the chosen action component is eliminated. This last phase can simply be a cut-off of the corresponding output or consist in a more gradual elimination in order to avoid catastrophic optimization events that may annihilate the performance of the policy.

6.2.2 Phase scheduling

Similarly to the first training phase where the convergence of the estimators is not easily predictable, the convergence of the metric in the evaluation phase may vary and depends on the choice of that metric as well as on the number of active action components. For this reason, these two phases of the loop do not have predefined durations in-terms of training epochs. Unless stated otherwise, a phase ends after a predefined minimal number of epoch and the compliance with a stability criterion. This ensures that indicators are converged while still avoiding useless computation of extra training epochs. In practice, this stability criterion is defined as the comparison between the absolute difference of two different Polyak averages ($\bar{\cdot}^{\alpha}$ and $\bar{\cdot}^{\beta}$) of the same metric with a threshold value (γ) that generally depends on the average value of the metric:

$$\left|\overline{\text{metric}}^{\alpha} - \overline{\text{metric}}^{\beta}\right| \overset{?}{\leq} \gamma(\text{metric}). \tag{6.2}$$

Commonly used values of α and β are 0.5 and 0.98.

6.2.3 Properly clipping the actor

Similarly to sensor sparsity, the actuator sparsity implementation later detailed uses a SGL placed downstream of the actor, as illustrated by figure 6.3 to dynamically clip the action. As, in the general case there is no need for computing gradients of the SGL tuning parameter α, a simpler version is adopted, that only consists of a vector of Bernoulli choice distributions. Thus, when the minimum number of epochs in the phase has been run and the stability criterion is verified, the algorithm proceeds to the following phase.

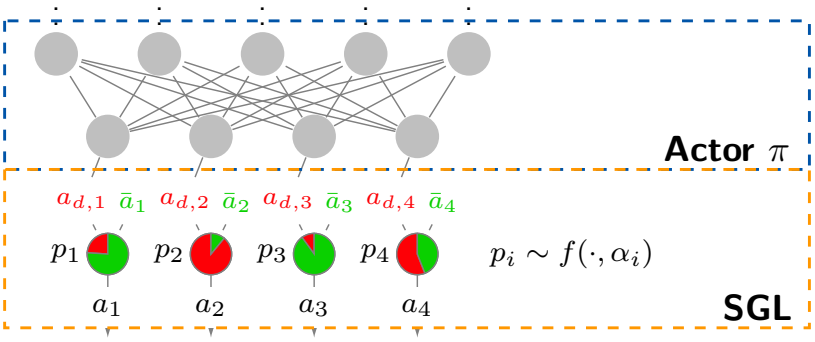


Figure 6.3: Illustration of the SGL structure generally used in the proposed implementations of the method.

Clipping is thus performed on the sampled action:

$$\underline{a}_{clip} = \underline{a}_{dense} \odot \underline{p} \left(+ (1 - \underline{p}) \odot \underline{\mu}_{sub} \right),$$

with $\odot$ being the scalar dot product, $\underline{\mu}_{sub}$ being substitution values (generally omitted since $\underline{\mu}_{sub} = 0$ in most of the cases) and $\underline{p}$ being the gate value, sample so that:

$$p_i = f(\alpha_i, u_i) = \begin{cases} 1, & \text{if } u_i \geq \alpha_i \\ 0, & \text{otherwise} \end{cases}$$

where $\underline{u} \in [0, 1]^{n_{act}}$ is a random vector and $\underline{\alpha}$ is a trainable vector that sets the probability for the gate to be open, as shown in figure 6.3.

To properly reflect clipping during the policy optimization phase, provided or computed values of μ and σ must be altered. As for clipped components there is no real sampling, likelihood functions computed in the actor loss should be corrected. First μ is clipped the same way the action is: $\mu_{clip} = \underline{\mu}_{dense} \odot \underline{p}$. Here $\underline{p}$ should not be resampled, the values of the gates during the roll-outs must be recorded and "replayed". In cases where the clipping is deterministic, this gate value storage is of course not needed. Second, the standard deviation vector σ is also biased on clipped components. The Gaussian log-likelihood of action a given μ and σ reads:

$$\log L(a|\mu, \sigma) = \sum_i \left[-\left(\frac{a_i - \mu_i}{2\sigma_i} \right)^2 + 2\log(\sigma_i) + \log(2\pi) \right].$$

Thus, for clipped components ($a_i = \mu_i = 0$), one adds $2\log(\sigma_i) + \log(2\pi)$ to the log-likelihood if no correction is performed. As σ is an optimized output of the actor in the case of the PPO-CMA,

this over- or under-estimation may lead to a collapse of the performance, since policy update are in that case based on biased likelihood values. To circumvent this issue, log-σ values are corrected in the following way:

$$\log \sigma_{clip} = \underline{p} \odot \log \sigma + (1 - \underline{p}) \odot \left(-\frac{1}{2} \log(2\pi) \right).$$

This way, when $p_i = 1$, the initial value of σ_i is preserved, but the $p_i = 0$ is replaced by a value neutral to the log-likelihood, that enables a faithful evaluation of likelihood one the activated components only.

6.3 Proposed ranking metrics

The second phase of the elimination loop consists in evaluating a metric used for choosing which actuator to eliminate. The current design is based on the evaluation of a scalar value m_i, $i \in [1, n_{act}]$ for each of the active actuators. The choice for the removal is done by ranking actuators according to (a Polyak averaged version of) this metric and removing the least performing action component $\arg\min_i m_i$. In the following, multiple metric options are detailed, discussed and compared.

6.3.1 Action norm

The first candidate is the simplest of them all. It relies on the assumption that injected energy by the forcing is somehow proportionally linked to its impact on the flow. Thus, quantifying by a norm the forcing action appears relevant as ranking indicator. Thus ranking metric m_i reads:

$$m_i = \mathbb{E}_{s \sim T, a \sim \pi} \left[||a_i||_2 \right] \quad \forall \ i \in [1, n_{act}].$$

Here the choice of an L_2-norm is rather arbitrary and can be discussed. This metric is simply computed on the roll-out data, no evaluation phase is then required. This metric does not require to train extra NN structures either.

6.3.2 Clipped agent training

The second proposed metric relies on a "what-if" analysis, where all the effects of clipping are estimated. This implementation is based on an actor-critic RL structure and leverages value function estimations as actuator ranking method.

In addition to the current agent, trained in a standard fashion, n_{act} extra clipped agents are spawn upon the beginning of the second phase of the elimination loop as illustrated by figure 6.4. Each if these n_{act} agents is clipped on a different action component, initialized with the parameters (biases and weights of both the main policy π and value function V) and trained separately on the environment during this second phase.

Upon convergence of each of these agents, a Polyak average of their value functions V^i is compared. The metric m_i reads:

$$m_i = \mathbb{E}_{s \sim T, a \sim \pi^i} \left[V^i(s) \right] \quad \forall \ i \in [1, n_{act}],$$

where π^i and V^i are respectively the policy and value function of the i^{th} agent that assesses the effects of clipping the i^{th} action component. Here m_i is the average expected return over the states visited if the i^{th} were to be clipped and the policy π^i converged.

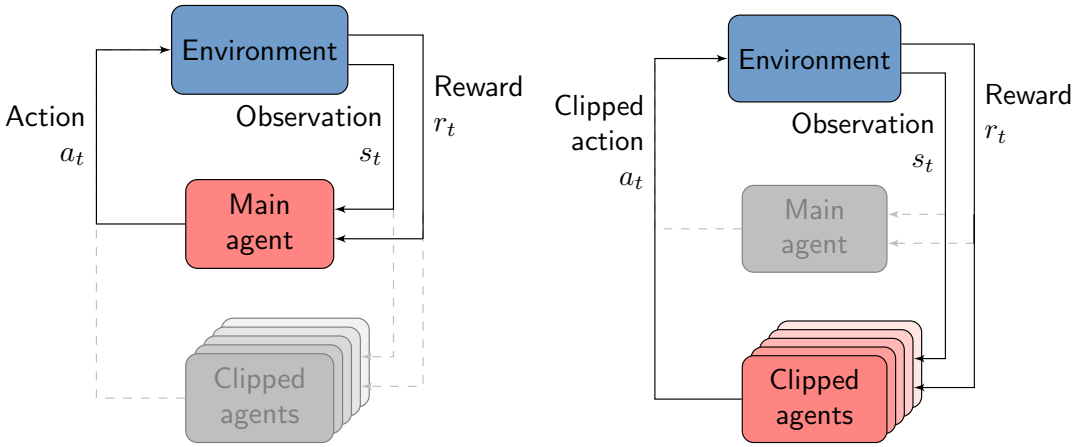


Figure 6.4: Schematics of the clipped agent training method. (left) Training of the main agent. (right) Training of all the clipped agents. There may be multiple parallel environments. Clipped agents are trained either sequentially on one environment or in parallel if possible.

The elimination phase is immediate since one only needs to transfer the parameters (clipping included) of the clipped agent corresponding to the eliminated component to the main agent. The following training phase is also in theory much reduced since the previous evaluation phase optimizes the clipped agents.

Here the evaluation of m_i requires the convergence of n_{act} agents (policy and value function), which may represent an important overhead as well as an increased memory consumption to store the parameters of the clipped agents.

6.3.3 Value function estimation

This next candidate metrics also relies on a "what-if" analysis performed on the value function, but this time without converging a clipped actor. Thus, as illustrated by figure 6.5, there is no need for n_{act} extra agents but only their value function estimators (or critic) V^i. These are trained on runs performed using the main actor π clipped on the corresponding action component. In that case the metric is fairly similar, except that actions are sampled on a different actor:

$$m_i = \mathbb{E}_{s\sim T, a\sim \text{clip}(\pi,i)}\left[V^i(s)\right] \quad \forall\ i \in [1, n_{act}],$$

where $\text{clip}(\pi, i) = (1 - \delta_{i,j})\pi_j$.

Contrary to agent clipping, the metric evaluation is performed using buffered data. Since the actor π is not updated during this phase, data collected on dedicated clipped runs on the environment remains *on-policy* and thus suitable to train the value function estimators all along an evaluation phase. Again contrary to the previous metric, both elimination training phases cannot be skipped since the policy π must be re-converged during and after each elimination.

For these last two metrics, the choices are simply justified by the fact that the value function is the best possible approximation of the expected return, that quantifies the compounded performance of the policy in time.

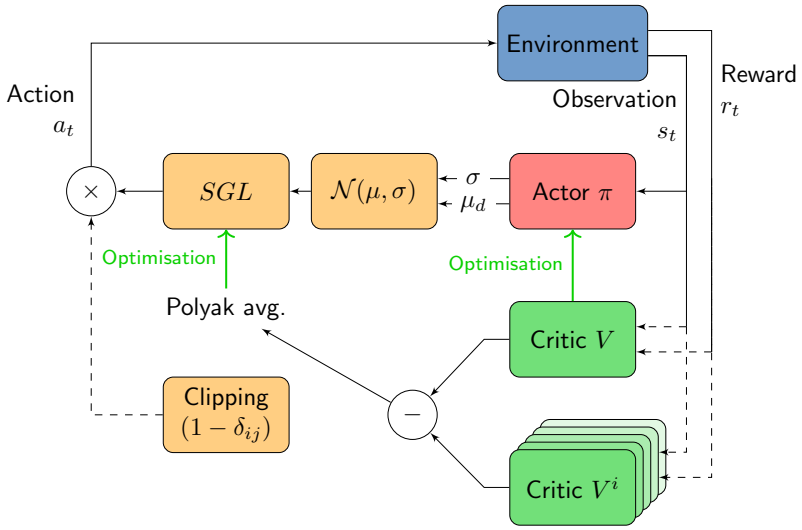


Figure 6.5: AS-PPO-CMA agent structure with value function estimation. The second training phase see the alternation of policy π and main critic V training with specific training for each critic V^i. The third phase focuses on tuning the SGL gate opening probability on the eliminated action component.

6.3.4 Mutual information estimation

This next metric is, similarly the previous ones, also deeply rooted into reinforcement learning hypotheses, and more precisely to the Markov Decision Process underlying the control problem. This metric proposes to rank actuators' actions a_i according to the mutual information $I(\Delta s, a_i - \mu_i)$ they share with the state dynamics $\Delta s_t = s_{t+1} - s_t$. Here μ_i is the i^{th} component of the best action $\mu(s_t)$ (one of the outputs of the actor neural network) and a_i is the i^{th} component of the sampled action: $a_i = \mu_i + \sigma_i \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, 1)$ and thus here $a_i - \mu_i = \sigma_i \varepsilon$. This choice is discussed later on.

Mutual information measures the dependence between two random variables, by giving the amount of information the observation of one variable gives about the other. I is 0 if both variables are independent. Given X and Y two random variables, one can write their mutual information as:

$$I(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

with H being the entropy of the random variable. This way, using $I(\Delta s, \sigma_i \varepsilon)$ as an indicator of the actuator "importance" relies on the hypothesis that the flow partial dynamics $s_{t+1} \sim T(\cdot|s_t, a_t)$ complies well enough to the Markov property so that s_{t+1} contains all the relevant information about the forcing effect of $a_{t,i}$. The heuristic is that, if a given action component is "useless", it won't notably influence the state dynamics, contrary to an "important" action component that will "drive" the dynamics.

In the latter case, the knowledge of such an action will reduce the uncertainty (H) about the state transition far more than in the first case.

Both entropy measures cannot be directly computed using the collected data. However, expressed as a Kullback-Leibler divergence $I(X, Y) = D_{KL}(P(X, Y)||P(X) \otimes P(Y))$, the Donsker-Varadhan variational formulation [65] enables to compute a neural-network estimator of I:

$$I(\Delta s, \sigma_i \varepsilon) = \sup_{\phi \in \mathcal{M}_b(\Omega)} \left(\mathbb{E}_{P(\Delta s, \sigma_i \varepsilon)} [\phi] - \log \left(\mathbb{E}_{P(\Delta s) \otimes P(\sigma_i \varepsilon)} \left[e^{\phi} \right] \right) \right),$$

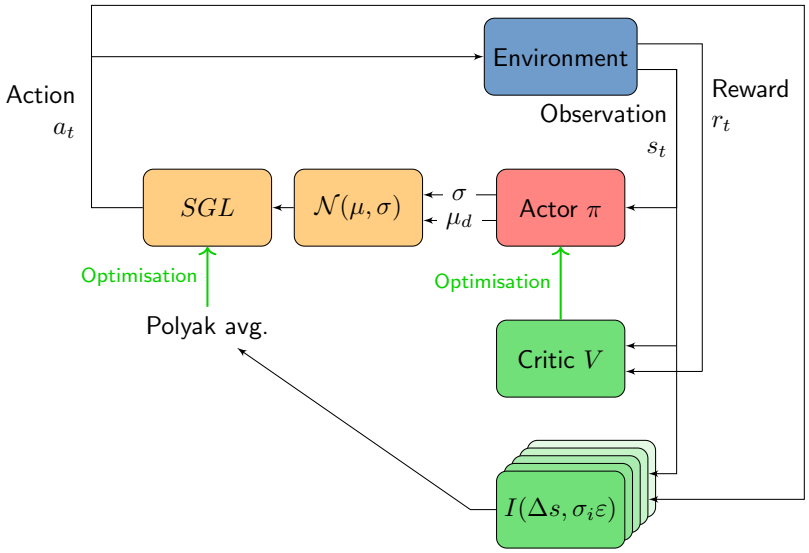


Figure 6.6: AS-PPO-CMA agent structure with mutual information estimation. In that case, both first and second training phases may be performed simultaneously since mutual information estimation can be performed "on-the-fly" without any extra data collection roll-out.

where Ω is the sample space of $(\Delta s, \sigma_i \varepsilon)$, $\mathcal{M}_b(\Omega)$ the set of all bounded measurable functions of Ω. Thus the function ϕ can be embodied by a neural-network and be trained to maximize the argument of the supremum as shown. This method has been successfully used by Belghazi *et al.* [15], Hjelm *et al.* [97] and numerous other studies in the domain of image classification. As illustrated by figure 6.6, each mutual information $I(\Delta s, \sigma_i \varepsilon)$ is estimated by a neural network (one per action component) and a Polyak average is run on these quantities to ensure stability throughout training.

The choice of $\sigma_i \varepsilon$ instead of a_i is motivated by the need to avoid spurious correlations. Action components a_i, indeed become increasingly correlated with each other (and deterministic) as the policy converges. Thus, one may misinterpret a strong mutual information $I(\Delta s, a_i)$ as a sign of importance of component i, whereas it may simply be strongly correlated with component j (for instance in an extreme case: $\mu_i(s_t) = \mu_j(s_t) \ \forall \ s_t$) that in turn is of (real) importance for the transition dynamics. One may write the control action a_i as $\mu_i + \sigma_i \varepsilon$ where μ_i is the major source of potential spurious correlation. Considering that σ_i is empirically rather constant with respect to s_t at a given epoch, $\sigma_i \varepsilon$ has been chosen instead of the "full" control action a_i as input for the mutual information.

As the evaluations and optimizations of the mutual information estimators are performed with *on-policy* data, the evaluation is expected to require a reduced number of training epochs compared to previously introduced methods.

6.4 Comparison of the metrics

For each of the proposed metrics introduced earlier, batches of test cases have been run on the KS environment. Similar configuration options have been used, notably concerning minimal convergence times and stability threshold, to allow for the fairest comparison possible. For all the metrics, batches of 50 to 200 test-cases have been run. On this test environment having 8 actuators, a systematic study has been performed, where all 255 actuation configurations have been evaluated. Each of

these layouts has been assessed using 5 test cases trained over 4000 epochs. Such an exhaustive study is only possible here because of the low computational cost of running the environment.

It has been found that the optimal layout evolution is complex and incompatible with a one-by-one elimination strategy: for a given number of actuators n_{act}, the best layout is not necessarily a subset (a "child") of the optimal layout for $n_{act} + 1$ actuators. Additionally, for a given n_{act}, the notion of "best layout" is subject to caution since several sub-optimal layouts display performances almost equal to the optimal. Overall, layouts with actuators evenly spread out in the domain demonstrate the best performances, without significantly differing from each others.

These aspects make the comparison of the exhaustive ablation study with the results using the elimination metrics rather complex. Thus it has been chosen to compare ensemble averages over the test batches rather than discussing slightly sub-optimal choices.

6.4.1 Control performances

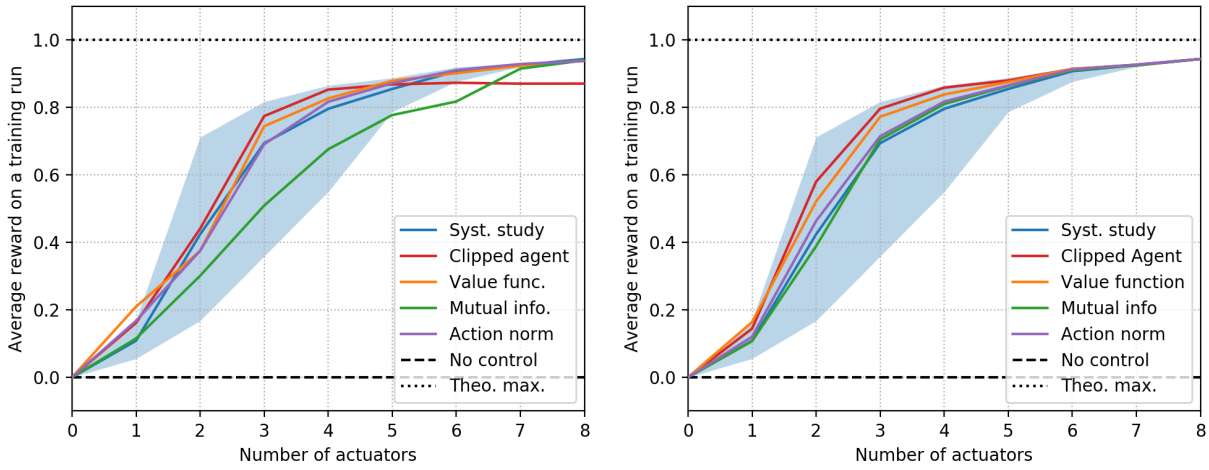


Figure 6.7: (left) Average training performance with respect to the number of active action components for each of the metrics. The results of the proposed metrics are to be compared with the ones of the systematic study (blue line). The blue shaded area represents the performance envelope of all possible layouts. (right) Expected performances of the proposed methods after an extra fine-tuning phase of the agents compared to the systematic study (blue line).

Figure 6.7 illustrates the average performances obtained by the different proposed metrics, looking directly at the policies obtained after eliminations (left-hand side graph) or after an extra fine-tuning of the policy on the sparse actuator layout (right-hand side graph). Considering this first graph, it can be noticed that except for the clipped agent (red line, this remains to be explained) all methods perform as well as the systematic study on large numbers of actuators. This may simply come down to the fact that all possible 6 or 7 actuator layouts perform rather equivalently. One can also notice that performances of the mutual information metric quickly drop compared to the others which remain comparable or sometimes better than the average of the systematic study. At last, the clipped agent, value function and action norm metrics provide performances with 1 actuator that are better than the theoretic maximum computed via the systematic study. One can assume that 1-actuator agents obtained by the elimination method take advantage of a broader exploration of the state-action space thanks to training phases with more than one actuator compared to the agents converged in the systematic study.

Considering the second graph where scores of the metrics are computed considering the performances of the obtained layout in the systematic study, both the clipped agent and value function metrics seem to clearly provide better performing layouts than the others.

6.4.2 Selectivity and computational costs

In addition to the obtained performances with sparse layouts, selectivity is another key performance indicator. Here selectivity is assessed as the observed selection frequency of a given action component (in the sense that the elimination process preserves it). Figure 6.8 compares these frequencies for the proposed metrics and for 1 to 4 active actuators.

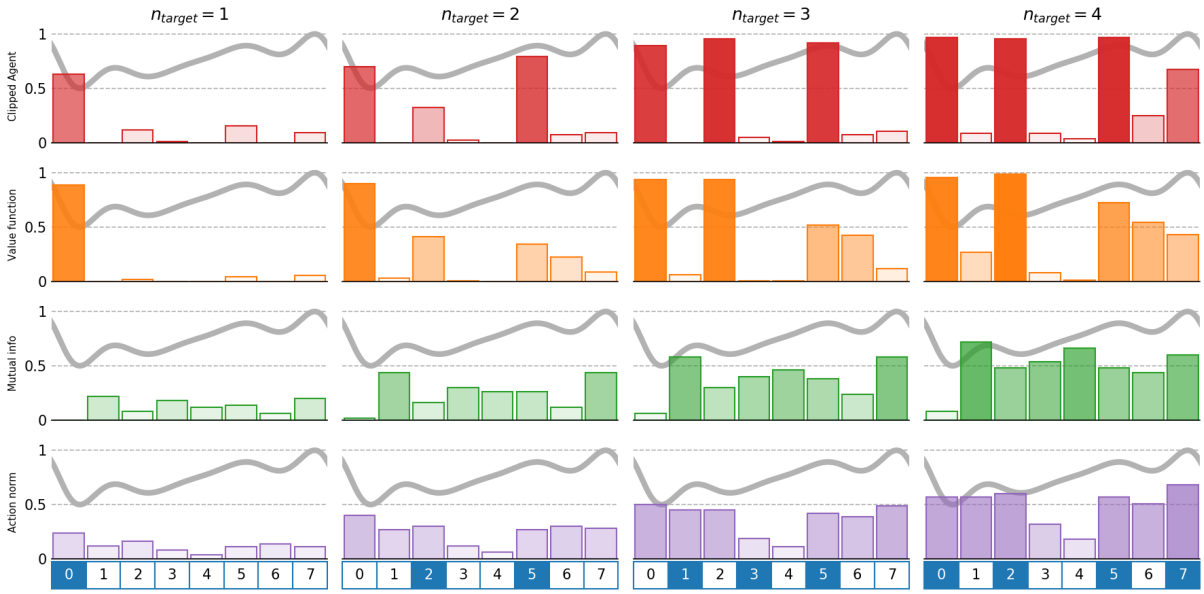


Figure 6.8: Frequency histograms of the obtained layouts (bar plots) for the clipped agent (red bars), the value function (orange bars), the mutual information (green bars) and action-norm (purple bars) metrics. These can be compared to the best layout from the systematic study (blue numbered boxes). Fixed point E_1 has been plotted in the background as a reference.

Starting from the bottom graphs, both the action norm and the mutual information metrics appear rather indiscriminate. The action norm still presents a small trend for eliminating actuators 3 and 4 before the others, whereas the only noticeable selective trend for the mutual information concerns the elimination of actuator 0. Concerning the agent clipping and value function metrics, they display very similar behaviors with early eliminations of actuators 1, 3, 4 and, to a lesser extent 6 and a preservation of actuator 0 as the last active component. These trends show the selectivity of these metrics and match also quite well with the best layouts according to the systematic study (refer to the blue boxes at the bottom of figure 6.8), even though they comply with the constraints of sequential eliminations in terms of possible layouts.

These trends are further confirmed by figure 6.9 (left) showing the observed layout entropy with respect to the number of actuators. The entropy H of the layouts is computed as follows:

$$H = \sum_{l \in \text{layouts}} f_l \log\left(f_l\right),$$

where f_l is the observed frequency of a given actuator layout over the batch. For all the metrics the entropy first grows significantly during the first two eliminations. But from the third choice onward, the discrepancy in selectivity between the clipped agent and the value function metric on the one hand and the action norm and the mutual information metric on the other hand is further confirmed by a decrease in entropy for the first metrics whereas the others see their entropy follow the trend of a random selection of actuators (blue line).

Figure 6.9 (right) compares the ensemble averaged costs of eliminations across the metrics. This cost is evaluated as the number of epochs between the start of the metric evaluation and the end of the policy adaptation phase. The last actuator elimination (from 2 to 1 action components) appears as the most expensive one. This is explained by the slower convergence of control performances after this elimination which causes the policy adaptation to last longer before complying with the stability criterion. This correlates well with the slow convergence of 1-actuator layouts of the systematic study (not shown). Surprisingly, the clipped agent and the value function metrics, which require the convergence of one or two types of *off-policy* estimators, do not require extensive numbers of epochs in-between eliminations, whereas the mutual information metric estimators seem to converge with more difficulty despite being *on-policy*.

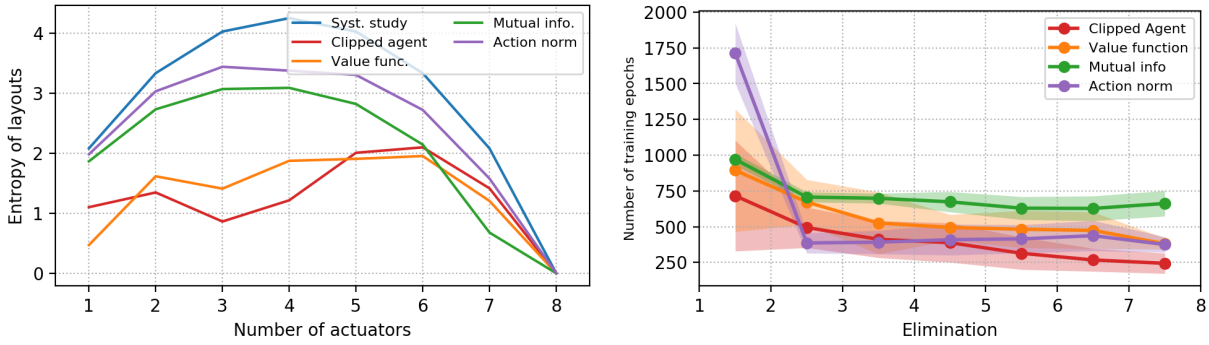


Figure 6.9: (left) Measured layout entropy with respect to the number of actuators. Values for the different metrics are to be compared with the maximum value achievable by randomly sampling layouts (blue line), corresponding to the systematic study. (right) Comparison of ensemble-averaged durations of eliminations across the proposed metrics. Solid lines represent averages whereas shaded areas account for standard deviation on the batches.

6.4.3 Application to the NACA case

Parts of this section are drawn from Paris *et al.* [175]. The proposed elimination algorithm has been applied to the previously introduced NACA airfoil flow, using the value function metric and for multiple values of the angle of attack α (12°, 15° and 20°). Figure 6.10 synthesizes the results of the sparsification on the cases with $\alpha = 15°$. Since performances follow similar trends for $\alpha = 12°$ and $\alpha = 20°$, the sparsification process is discussed for $\alpha = 15°$ only. One can distinguish two different trends in the performances. First, from 10 to 4 actuators, only the average values of both drag ($\overline{C_d}$) and lift ($\overline{C_l}$) coefficients (refer to lower left and right graphs of figure 6.10) are significantly impacted by the elimination of actuators, respectively seeing an increase and a drop of their value as the number of actuators is reduced. Standard deviations of these coefficients (on which the reward is based), quantifying the steadiness of the mechanical loads, remain rather constant. As the number of actuators further decreases, from 4 to 1, both average coefficient values pursue and

amplify their previously described evolution. Standard deviations significantly increase, denoting an expected decrease in overall performance. For the vast majority of the data points, the control yields significantly better performances compared to the non-controlled flow (denoted as "baseline" on figure 6.10).

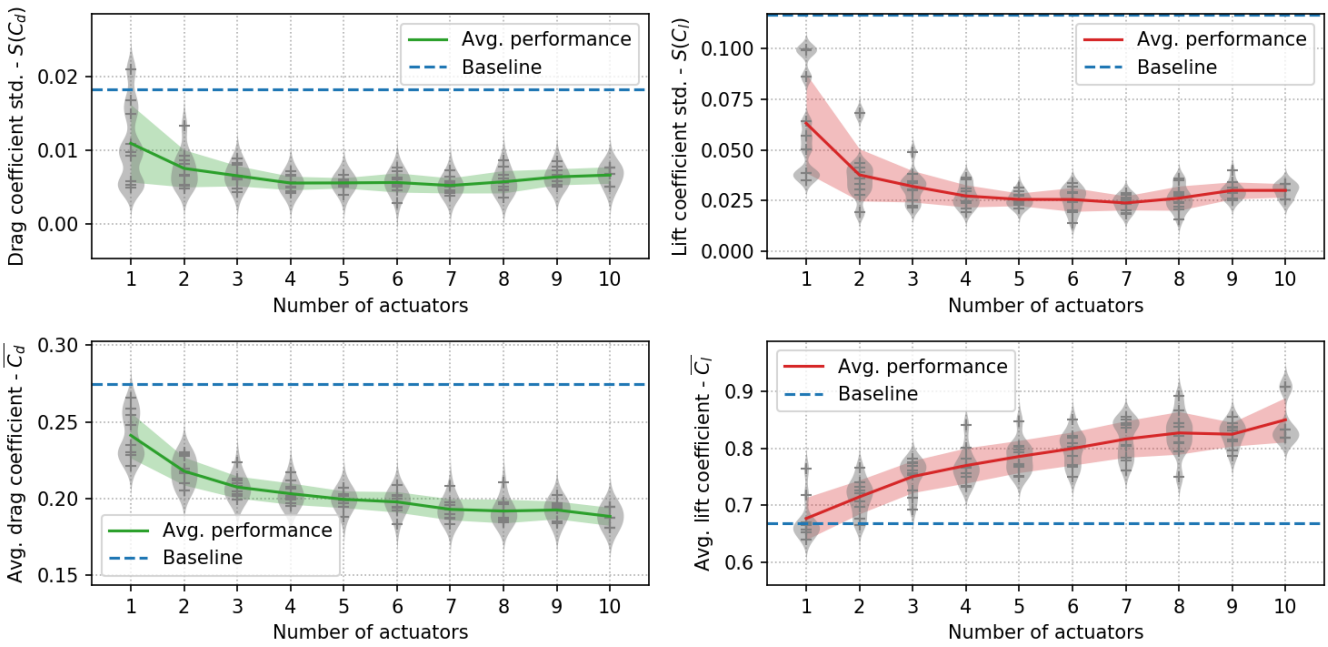


Figure 6.10: Averaged training performance indicators with respect to the number of active action component for $\alpha = 15°$. Data points are denoted by gray "plus" signs. Ensemble averages are computed on these data points (batch-size 5). The blue dashed line represents the uncontrolled (baseline) performance, solid lines denote the evolution of the ensemble average and shaded areas illustrate the standard deviation across the batch. All indicators are computed on the last 40 control steps of a training run. (Upper left) Evolution of the standard deviation of the drag coefficient $S(C_d)$. (Upper right) Evolution of the standard deviation of the lift coefficient $S(C_l)$. (Lower right) Time-averaged drag coefficient $\overline{C_d}$. (Lower left) Time-averaged lift coefficient $\overline{C_l}$.

Figure 6.11 further describes the elimination choices with respect to both the angle of attack α and the number of active action components. In all cases, the algorithm chooses actuators consistently to the first analysis proposed in the end of section 4.3.1. With 5 and 3 components, eliminations for all studied angles of attacks display similar patterns, removing actuators 4 and 5, preserving other actuators with a slight advantage for upstream ones (0 to 3). For $\alpha = 15°$ and $\alpha = 20°$ and for a reduced number of action components (1, 2 and 3), the algorithm favors locations near the separation range, as one may intuitively expect. However, for $\alpha = 12°$ the remaining actuators' location differs from the separation range. This turns out to be more beneficial for this specific angle of attack.

The study of this test case has first shown that, except for $\alpha = 20°$, a nearly complete stabilization of the flow using reduced-amplitude control actions, could be obtained with the proposed layout and using the described reinforcement learning method. The proposed action sparsification algorithm has shown expected results while still allowing to significantly preserve control performances during the elimination phases. For reduced numbers of action components, the feedback provided by the sensors enables the policy to properly synchronize control actions with the remaining unsteadiness

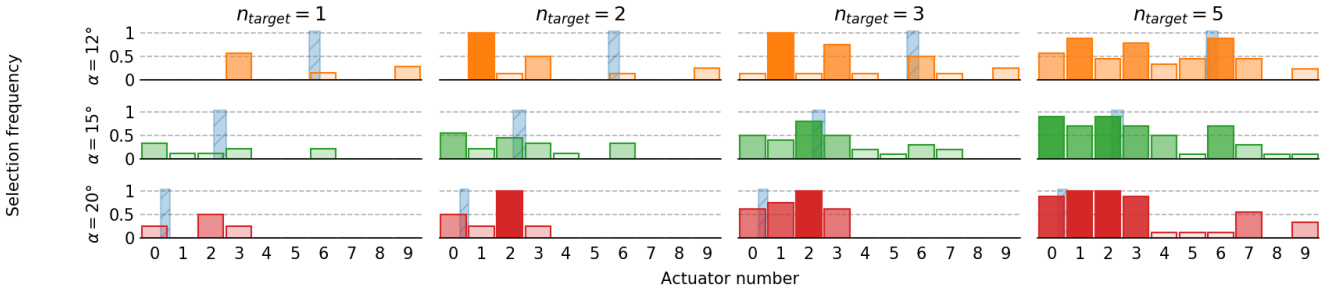


Figure 6.11: Frequency histograms of the obtained layouts (bar plots), for $\alpha = 12°$ (top row), $\alpha = 15°$ (middle row) and $\alpha = 20°$ (bottom row). The blue shaded areas correspond the range of evolution of the separation point. These frequencies are observed on the test batch of size 10, used in this study.

and thus, to reduce effective load variations on the airfoil.

6.4.3.1 Remark on the obtained closed-loop control laws

The airfoil case has been selected because of the rather simple control policies it yields (in most of the cases, a suction in the vicinity of the separation region), which makes it a good candidate to assess the behavior of our actuator elimination algorithm. As the control actions appear nearly steady during the late phase of control (whether it is with the full 10-actuator layout or the sparse ones), one may question the intrinsic closed-loop nature of the control law similarly to the discussion in section 4.3.2. Even though this question is not in the original scope of the present study, it is nonetheless an interesting side point that is briefly addressed in the following.

Closed-loop tests runs have first been performed for 5 and 2 actuators for all considered angles of attack, to be used as baseline values. Again here, control action sequences have then been re-implemented in an open-loop fashion on randomly reset cases. Table 6.2 presents the variation of various performance indicators at the end of the test runs. Results with 10 actuators, already presented in table 4.4 have been added for the sake of comparison.

Relative variations of the time-averaged lift coefficient are systematically positive, whereas the drag coefficient evolves negatively. These variations yet remain small compared to the absolute coefficient values. Standard deviations quantifying the steadiness of the loads on the airfoil (which constitute the actual control objective defined in the reward) are systematically increased when implementing control action in an open-loop fashion. The difference between open-loop and closed-loop can be seen for 5 and 2 actuators, where the closed-loop does not totally stabilize the flow. Where here it can be considered that for 10 actuators, the policy is close to an open-loop forcing, open-loop policies display a lesser efficiency for a reduced number of actuators, where the remaining load unsteadiness can be damped by an in-phase control action.

To conclude, the relative improvement obtained by using a closed-loop strategy over an open-loop one is significant here. But regarding absolute gains, they are more marginal. Therefore, it may be feasible to get satisfactory control performances on this case through a parametric open-loop control study (with constant or variable suction), where the number, action amplitude and sign, and actuator position would be varied. Yet, even for this simple flow configuration, such an exhaustive study would face the "curse of dimensionality" inherited from the combinatorial search of optimal layout (with, for instance, 252 different layouts having 5 active action components). Consequently, if carried out blindly with respect to the flow physics, this search would end up being significantly

Nb. act.	α	C_l	C_d	$S(C_l)$	$S(C_d)$	$S(C_l) + S(C_d)$
10	12	0.0%	0.0%	0.0%	0.0%	0.0%
	15	+2.1%	+1.5%	+91%	+6.8%	+70%
	20	+0.9%	+0.8%	+37%	+57%	+42%
5	12	+4.3%	+2.6%	+154%	+132%	+151%
	15	+2.9%	+2.1%	+120%	+98%	+115%
	20	+1.8%	+1.7%	+47%	+43%	+46%
2	12	+5.1%	+2.9%	+126%	+107%	+123%
	15	+4.6%	+3.4%	+89%	+86%	+88%
	20	+1.8%	+1.7%	+12%	+24%	+14%

Table 6.2: Comparison of the final (in the "stabilized phase") performances in closed-loop and open-loop conditions. Performance variations are measured as relative variation with respect to the closed-loop performance. Green colored figures indicate that open-loop control performs better on the metric than closed-loop whereas red colored ones state the opposite. Ensemble averages are computed on batches of 10 test runs.

more expensive than the present algorithm, even for this relatively easy-to-control flow.

6.5 Model-based model predictive control

In this context where the extra exploration required by the estimation of metrics represents large computational overheads, model-based approaches could take advantage of their model by running this exploration on their reduced-order model, without requiring extra costly runs on the full-state environment. An action-sparsity-seeking version of the PETS-MPC algorithm has been develop to this end. It relies on the training of the forced transition model $(s_t, a_t) \rightarrow s_{t+1}$ that is later used to assessed the value of clipped policies. As the policy is implicitly defined as the sequence of actions minimizing the estimated cost over a given time horizon, only minor changes are needed to estimate the value of clipped policies. A simple binary mask has been implemented downstream the action sampling object in the action optimizer. By alternatively setting its values to zero, one can simulate the behavior of a clipped policy on the reduced dynamics.

Yet here, the question of extrapolation, discussed about the full-state environment, re-emerges concerning the predictions capabilities of the model. This transition model is indeed trained only on *on-policy* data, meaning that there are no guarantees regarding its performance on out-of-distribution samples. And as clipped policies may lead to different regions of the state-space this issue may be critical for the estimation of the value of these policies.

Figure 6.12 shows the prediction accuracy of the models depending on the number of active components. Unsurprisingly models perform better on 8-actuator policies. This region of the state space has been "seen" during training and thus models only perform interpolation, whereas for clipped policies, the need for extrapolating the learned dynamics leads to decreased performances. Reusing the threshold introduced in section 4.1.3.2, the informative prediction horizon decreases from 30 to 40 control steps for 8-actuator policies down to approximately 10 control steps for policies having only 5-actuators. In the framework of the sequential elimination and assuming similar trends at other stages of elimination, it can be estimated that the accuracy of the model is divided by 2 when querying it for clipped policies (the prediction horizon with 7-actuator policies being around 20 control steps). In the current study, the value of clipped policies is then evaluated starting from

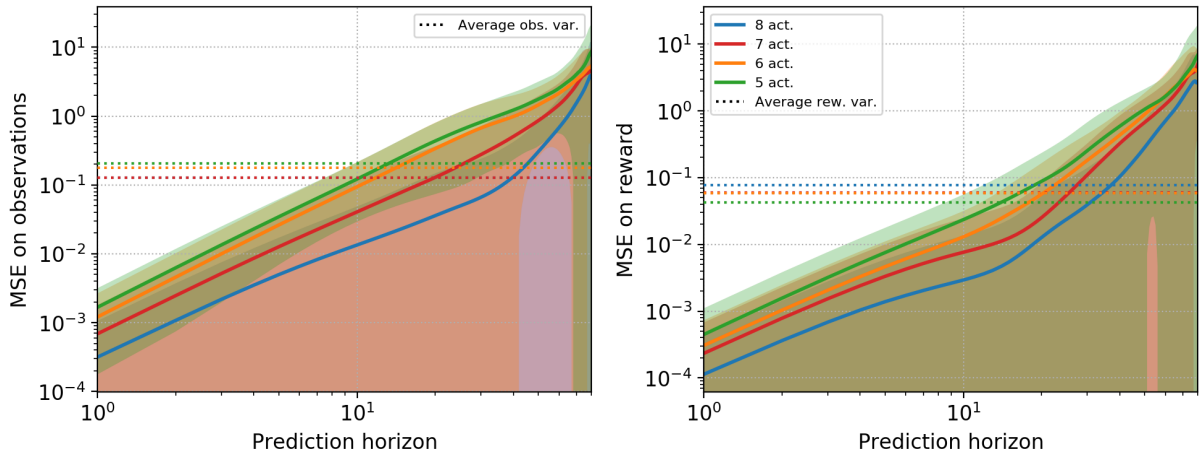


Figure 6.12: Prediction accuracy of 15 models trained on 8-actuator policies for a varying number of active components. Mean square error on the observation (left) and reward (right) prediction. Average variance of observations and reward (on the full-state environment) are plotted in dashed lines for comparison.

100 different observation vectors and on horizons of 10 control steps, thus remaining within the assumed accuracy horizon of the models.

Figure 6.13 gathers the performances of this method on different key performance indicators previously discussed. It can be observed that performances obtained without fine-tuning are noticeably below the ones obtained with previous model-free methods (refer to figure 6.7). But after fine-tuning, average performances are similar to what has been obtained with the clipped agent or the value function metric. This has yet to be explained, all the more that the observed mean squared error of the models does not drop after an elimination. This would be expected since eliminations alter policies and the distribution of observed states which in turn puts models out of their training domain.

Layout entropy is also similar to the results obtained with the two best performing model-free metrics. Concerning the average cost of an elimination conversely, much better performances are achieved. These still must be contrasted with the fact that model-based training epochs are much more computationally intensive (due to the control action sampling) than their model-free counterparts, especially on environments where action sampling and policy optimization make up the majority of the computation.

Figure 6.14 confirms the selectivity sketched by the layout entropy, with selection frequencies displaying a strong reproducibility and layout choices strongly correlated to the best layouts according to the systematic study. Overall these encouraging results call for more investigations on the matter, especially concerning ways to ensuring model accuracy when extrapolating the value of clipped policies. Hybridization with explicit, neural-network-based policies could also help reduce the cost of training epochs, by avoiding an optimization process at every control step, while still preserving the capacity to compute *off-policy* estimators.

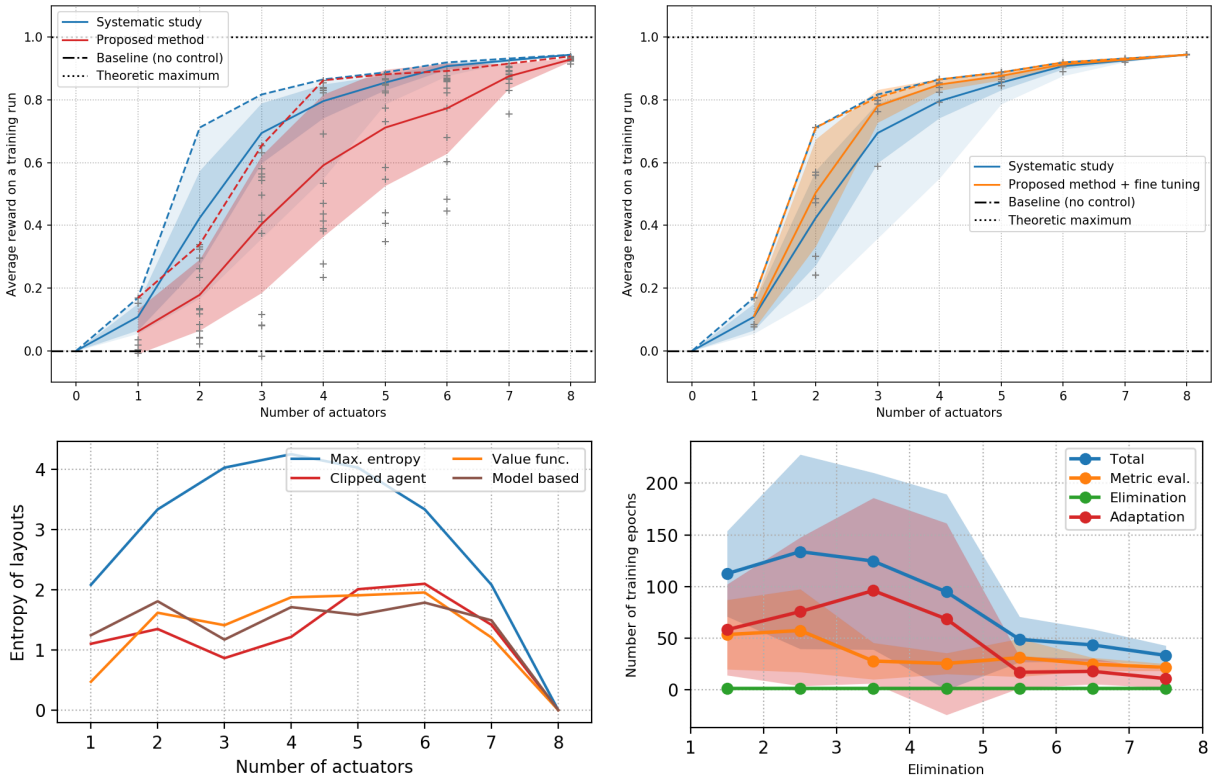


Figure 6.13: (top left) Average training performance with respect to the number of active action components (red line) compared to the systematic study (blue line). Shaded areas represent the standard deviation across the batches. (top right) Expected performances of the obtained layout and models (orange line) after an extra fine-tuning phase of the agents compared to the systematic study (blue line). (bottom left) Measured layout entropy with respect to the number of actuators (brown line). Values are to be compared with the maximum value achievable by randomly sampling layouts (blue line), corresponding to the systematic study and to the results obtained with both clipped agent (red line) and value function (orange) metrics. (bottom right) Comparison of ensemble-averaged durations of eliminations across the proposed metrics. Solid lines represent averages whereas shaded areas account for standard deviation on the batches.

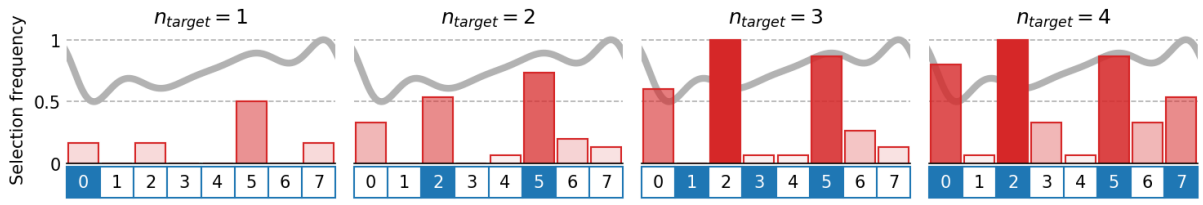


Figure 6.14: Frequency histograms of the obtained layouts (bar plots) for the action sparse model-based method. These can be compared to the best layout from the systematic study (blue numbered boxes). Fixed point E1 has been plotted in the background as a reference.

6.6 Synthesis

6.6.1 Discussion of the current results

The metrics and methods introduced in the current part, showed a varying efficiency regarding the issue at hand: reducing the number of required actuators, starting from a large actuation layout while preserving the control performance as much as possible. It is first important to state that these are preliminary results call for further work on the topic and the practical implementation of the stochastic estimators underlying these metrics. The current results are not definitive conclusions to the efficiency of a given metric. Overall current results suggest that both the clipped agent and value function metrics perform satisfactorily but represent a large computational overhead. It is worth noting that, on this test-case at least, the clipped agent requires on average less extra training epochs than the value function, which is counter-intuitive since the first has two interacting estimators to converge for each elimination whereas the second only converges the value function of a frozen sub-optimal clipped policy. Regarding the action norm, performances and selectivity are, as expected, relatively low, since this metric relies on a most likely spurious correlation.

Concerning the mutual information metric, there still remains an unresolved question concerning the balance between the ease of convergence of the mutual information estimators and the risk of inducing spurious correlations. The current choices of a single neural network estimator, having only the "random" component $\sigma_i \varepsilon_i$ of the i^{th} action component as input leads to low reproducibility and performance. Other options are considered such as having multiple estimators that are trained on different data batches and averaged in order to stabilize the estimate of the mutual information. Another option considers multiple actors that all converge their own mutual information estimators, taking the whole action component as input (i.e. $I(\Delta s, a_i)$ instead of $I(\Delta s, \sigma_i \varepsilon_i)$). Then, by ensemble averaging over all the actors' estimations, one could discriminate mutual information values due to a real causation, that would all present a similar value across all actors, from those due to spurious correlations, that would show varying values (i.e. a large ensemble standard deviation) across the actors. These ideas require a non-null coding effort to be properly implemented and tested, but they are worth trying, since the issue of *on-policy* estimation of a quantity that appears as *off-policy* could noticeably reduce computing times.

The model-based approach shows encouraging results on the current test-case both in terms of selectivity and performance. There are still unresolved questions about the adaptation of the model after an actuator elimination. Complementary studies should be led to better understand the impact of the change in state-action input distribution on the training of the neural-network and on the optimization process of the control action.

The issue of the metric's relevance is also central. For both the clipped agent, the value function or the action norm analyses, this question is trivially answered. Positively for the first two and negatively for the latter. This yet looks less obvious for the mutual information metrics as well as for the proposed model-based method. In both cases, it is a matter of extrapolation. Assuming the Markov property[2] and that the mutual information measured on the *on-policy* state-action distribution is representative of the one experienced by the clipped policy, the mutual information approach appears valid as long as low-variance estimators, providing faithful values for a moderate overhead can be computed. Similarly, the proposed model-based approach relies on the extrapolation capabilities of its reduced model. It is a well-known fact that neural-networks are very powerful interpolators but generally collapse when extrapolating. Thus considering methods to either estimate the degree

[2] refer to sections 2.2.1.1 and 7.4.

of uncertainty of the model (such as disagreement between models) and/or to improve extrapolation accuracy are part of the future work on this topic.

6.6.2 Perspectives

RL assumes (by default) a non-linear behavior of the system and the control policy. Yet concerning the system, its dynamics still contains some structure (continuity, some degree of differentiability, upper and lower bounds, energy conservation, etc.), inherited from the underlying physics it represents. Capturing (parts of) this structure and embedding it into the models and estimators would dramatically improve the training efficiency of the methods. The links with (sparse) symbolic regression [123, 57, 173] or spectral submanifolds [179] still reveal that this problem is notoriously tricky in high-dimension (and with scarce data) without a prior knowledge of the system (such as for the APHYNITY algorithm [264]).

Multiple actuator-elimination methods have been introduced in this chapter, resorting to either model-free or model-based RL algorithms, but all following the sequential elimination process introduced in section 6.2. This paradigm imposes two main constraints:

First, the initial layout defines the "envelope" of the search space. One could think of a more flexible paradigm that may move and/or add actuators instead of only removing these. While allowing for reaching layouts otherwise unreachable with the current paradigm, this would still pose some practical issues concerning all NN structures and data management pipelines that would have to deal with potentially unbounded and varying numbers of inputs and outputs. This may also break the halting guarantee of the process and may lead to unpredictable elimination stages far from the user target and unpredictable computational costs, as discussed in section 6.1.2. Considering these facts, the constraint of sequential elimination, that strongly reduces the possible outcomes is what makes the process less intractable.

Second, eliminating one actuator at a time also constrains the outcomes. If the algorithm were to eliminate more actuators simultaneously, two strategies could be considered, yet both seem inaccurate or unfeasible. One could assume a sort of linearity in the estimations, meaning that estimations from a single component clipping would be representative of multiple simultaneous clippings. As shown by figure 6.15, this hypothesis is invalid. For instance when comparing the rankings for the first two eliminations, components 4 then 1 would be the first to eliminate. Yet after the first elimination, actuators 3 and 6 come behind actuator 1 in the ranking. This change in the order of actuators can be explained both by the change in state distribution on which clipped value functions are evaluated and by the policy adaptation that converges again to compensate for the loss of an action component. Physical neighboring effect is also not to be excluded. Two actuators located near one another may fulfill the same function. Thus the elimination of one may boost the importance of the other that it may replace. In other words, the ranking at elimination step 1 is likely not the average observed (and assumed optimized) elimination order. The second strategy would consist in performing estimations for each of the $\binom{n_{act}}{n_{act}-n_{simult}}$ possible combinations, n_{simult} being the number of simultaneously eliminated components and n_{act} the current number of active components. This generally becomes intractable passed $n_{simult} = 1$, except if the evaluation cost is orders of magnitude cheaper than one training epoch. This may be the case for model-based RL whose goal is exactly to provide these computationally cheap estimations.

As of this study, the proposed methods remain out-of-reach for larger and more expensive flow control environments. Efforts regarding the sample efficiency and the quality of the metrics is part of future works. Hybrid model-based/model-free approaches may be relevant to this issue by

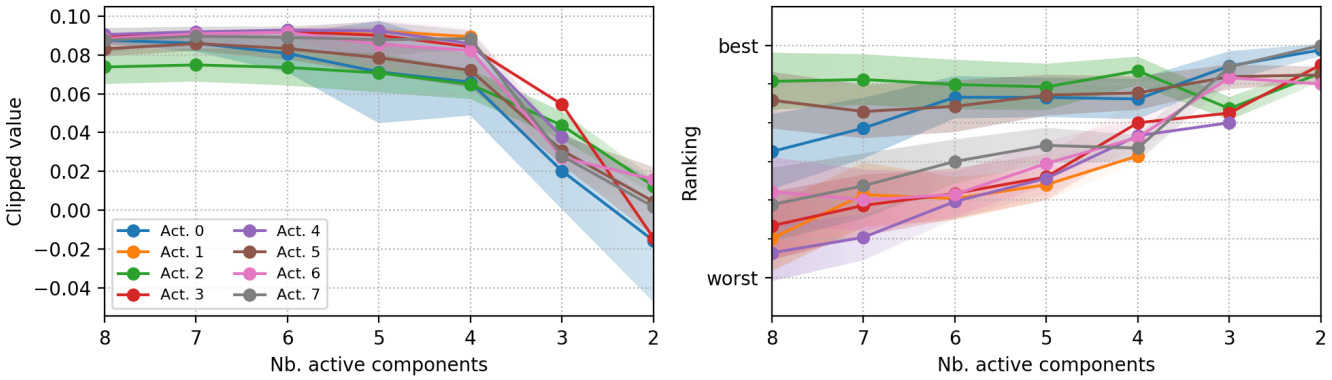


Figure 6.15: Evolution of the ranking of actuators during the elimination process using the agent clipping metric. (left) Ensemble averaged clipped value estimations at the end of each metric evaluation phase. The higher the clipped value, the lower the importance of the corresponding actuator. (right) Ensemble average ranking of actuators deduced from the clipped values. The worst-ranked actuator is eliminated during the following phase.

bringing the best of both worlds: cheap, low-fidelity approximations from model-based methods combined with the accuracy of the convergence of principled methods from model-free algorithms. Efficient means of arbitration between cheap and error-prone extrapolation on the one hand and costly but accurate exploration on the other, could also help reduce the computational cost of actuator selection while preserving performance.

Chapter 7

Open challenges

This chapter discusses multiple issues that arose repeatedly across the test-cases. These issues are generally not encountered on more traditional and simpler test-cases such as the ones commonly used to qualify and compare state-of-the-art RL algorithms. Similarly to sensor and actuator layout optimizations, these may not be specific to flow control in the sense that most of the scaling-up applications of RL are likely to display similar characteristics. The discussion aims at taking a step back on these general problems and proposes general-purpose mechanisms and ideas to help solve these obstacles and provide more sample-efficient methods.

Contents

7.1 Exploration noise

Most of the efficiency of RL is about the optimization of the exploration-exploitation trade-off. With *on-policy* algorithms, exploration is performed via different methods, only in the vicinity of the trajectories normally explored by the current policy. Most of these methods consider additional control action noise. These exploration strategies are directly constrained by the need for faithful stochastic estimators.

7.1.1 Computing low-variance, accurate estimators

As described early on, the improvement of policies relies on optimizations by gradient descent based on losses themselves computed using stochastic functions. As these functions cannot be directly computed because of a general lack of knowledge about the environment dynamics, they need to be estimated. In turn, for the optimization to properly deliver, these probabilistic estimators should be unbiased and display the lowest variance possible.

Some estimators such as the one for the value function V are the most precise when exploration is the lowest (i.e. $\sigma = 0$ if using the PPO-CMA algorithm). Conversely, others such as the advantage function A rely on this exploration to provide the optimization direction for policy updates. The latter provides finite-difference-style estimations[1] of potential performance improvements and in that case, the parameter driving exploration can be directly related to the average step size used to perform these finite-difference estimates.

Generally one needs a non-null but yet as low as possible exploration in order to compute the most accurate stochastic estimators. Yet from a practical point of view, low exploration empirically means slow improvement and thus long and costly convergences. On the other hand, a too large exploration is synonym for noisy estimators that cannot provide trustworthy gradients, which may again either slow down convergence or even freeze it.

The variance of the estimators can be explained by two main factors. For action-dependent estimators such as the advantage function, noisy control actions will obviously cause high-variance. Second, and this is valid for all types of estimators, a noisy action may alter significantly the distribution of visited partial states s. This distribution of states may be of high variance, and more importantly it may be completely different from the one expected when using the policy without exploration noise. This would mean that the "exploration policy" and its noiseless version explore different regions of the state-action space and thus estimators drawn from one agent provide totally irrelevant gradients for the other. For instance on turbulent-transition-prone environments, additional exploration noise may trigger this transition, whereas it would not occur with the noiseless policy, leading the flow into completely different flow regimes having their own dynamics.

At last, the batch size, i.e. the amount of data on which these estimators are computed, is also an important matter. This batch size is generally driven by the roll-out length in number of control steps and by the number of roll-outs performed by epoch. Similarly, small batch sizes may lead to high-variance estimators, irrespective of the "quality" of the data samples, whereas large batch sizes may represent a significant computational overhead, that in turn translates to slower convergence.

[1] The advantage estimates the performance the policy would have if a given action a were to be taken instead of $\mu(s)$ with partial observation s.

Consequently there is a trade-off to be found for an efficient exploration as well as a speedy convergence. This trade-off depends on both the environment and the training algorithm, less chaotic environments being more resilient to large exploration noises and simpler estimators being easier to converge.

7.1.2 Levers

Concerning *on-policy* training algorithms, multiple strategies have been tested in order to optimize further the trade-off previously introduced. Using the PPO-CMA as base algorithm, some are introduced in this section.

PPO-CMA itself is a first strategy used to solve this issue, in the sense that in comparison with the PPO which is one of the go-to methods for *on-policy* RL, the PPO-CMA proposes a dynamic scheduling of the exploration noise variable σ by making it a trained output of the policy as well as the best action μ. Conversely PPO considers a fixed value for σ, preemptively chosen by the user. Empirically when using the PPO-CMA, one observes than σ decreases gradually then stabilizes around values much lower than the ones of similar successful training attempts using the PPO. Convergence is also generally faster with the PPO-CMA.

7.1.2.1 Initial σ value

PPO-CMA drives σ along the training but an important parameter conditioning the length of the convergence is the initial value of the exploration noise. As said, σ is seen to be generally decreasing along the convergence. Thus to a certain extent, forcing a lower initial value may cut this duration in the case where a significant part of the policy improvement consists in making it more deterministic (i.e. the exploration noise is initially too large and slows down convergence because it explores regions irrelevant to the control). Conversely a low initial value may simply prevent the agent from a proficient exploration and in turn could freeze training.

The impact of this initial value has been assessed using an additive bias on the $\log \sigma$ (thus multiplicative concerning σ) output of the actor neural network. Initialization of the average σ value is simply done by running a first epoch, and then tuning the bias in order to match the target value, under the assumption (generally verified) that this tuning reaches its target on the following epochs and that there is no feedback effect requiring a multi-step tuning.

Figure 7.1 illustrates the impact of such an initialization on training performances for different environments and initial values. First, on the KS environment, one can notice that the only significant impact on performance is obtained for a very low initial σ value. Here, as the control action ranges $[-1, 1]$, an initialization of $\log \sigma$ at -3, induces an uncertainty of around 5×10^{-2} or around 2.5% of the action range. In that case, training is much slower than with a larger initial value or no initialization at all. This can be explained by the fact that, with 8 action components, the sampling distribution peaks very sharply as $\log \sigma$ decreases. Around 8 action components the expected log-likelihood decays nearly linearly with both an increase in the dimension and a decrease in $\log \sigma$. Thus distributions are closer to a Dirac distribution that smooth heavy-tailed Gaussian ones. This in turn flattens the gradients and freezes training.

The same study on the cavity, using 5 actuators this time shows different conclusions. Here the smaller the initial σ, the better the training. This time the effects yielded by noise control actions on the environment seem prevalent over the issue of the collapse of distributions at play on the KS test-case.

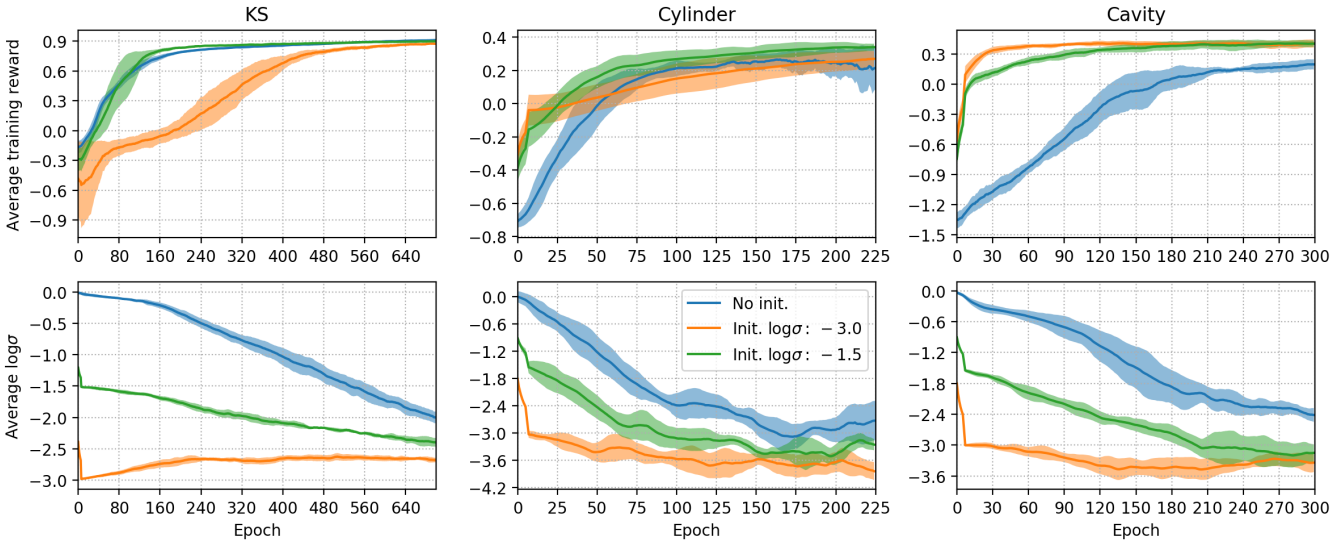


Figure 7.1: Comparison of the impact of the initial $\log \sigma$ value on performances on different environments. Top graphs illustrate the evolution of the average training reward, where bottom graphs show the evolution of the $\log \sigma$ optimized by the PPO-CMA agent. From left to right, studies respectively run on the KS, the standard cylinder environment and the cavity test case.

At last on the cylinder test-case, with only 1 action component, the initialization does not yield any significant effect on the efficiency of the training. The environment seems resilient to action noise (considering for instance a Lyapunov exponent, not developed here) and the reduced action space dimension (only 1) makes the slope of the linear decay of the log-likelihood much smaller. In some sense, having one action component instead of 5 or 8, inhibits the effects of the initial $\log \sigma$ value and thus the need to tune it precisely.

7.1.2.2 Random environment reset scheduling

At the beginning of training, the agent samples almost fully random actions. As discussed previously in section 4.4.3.1, some environments may display both delayed, long-term dynamics stating the question of credit assignment, but also a significant noise sensitivity that may amplify noisy control actions and build up chaotic states. On such cases, running long training episodes is mandatory in order to have a chance to "visit" relevant states, but they may provide noisy and uninformative data samples at the beginning of training coming from the end of the training trajectory. A simple idea to circumvent this issue is to run shorter episodes after which environments' states are reset with a decreasing probability along training via a scheduling. Not resetting environments' states corresponds to starting the following roll-out on a "pre-controlled" state and enables to visit states "further" from non-controlled initial states within the allocated control step budget (i.e. the roll-out length).

This way, at the beginning of training, only partial states for which potential chaotic fluctuations that have had a reduced number of steps to develop are used to train the agent. Then, once the latter has improved and reduced its exploration noise levels, the scheduling may enable to artificially simulate much longer training runs by stitching the data samples coming from two or more consecutive roll-outs without environment reset in-between. The case of the cavity, displaying these precise characteristics, has been used as a first test for this idea. Using the control setup

introduced earlier (training episodes of 250 steps), the effect of the scheduling of probability of resetting the environment state at the end of training roll-outs has been assessed.

Figure 7.2 compares the effect of the scheduling introduced in section 4.4.3.1 (P(env. reset) decreases from 1 to 1/20 between epochs 700 and 900) with other possible choices: a fixed reset probability set to 1/20 from the beginning of the training (green lines) and a systematic reset of the flow state (orange lines). Here the reference setup displays better value function losses and a lower average $\log \sigma$ than the two other options. Its training performances are slightly better concerning the metric m_3 (introduced previously and quantifying the upstream fluctuations) but match the ones of the batch having a systematic 1/20 environment reset probability (at the exception of a slightly faster initial improvement until epoch 500). Surprisingly, while being systematically better than the agents trained with a systematic reset (orange batch) on training performances, evaluation performances show that, on the metric of interest m_8 (quantifying downstream fluctuations), systematic reset environments tend to provide slightly better performances on 4000-step test-runs.

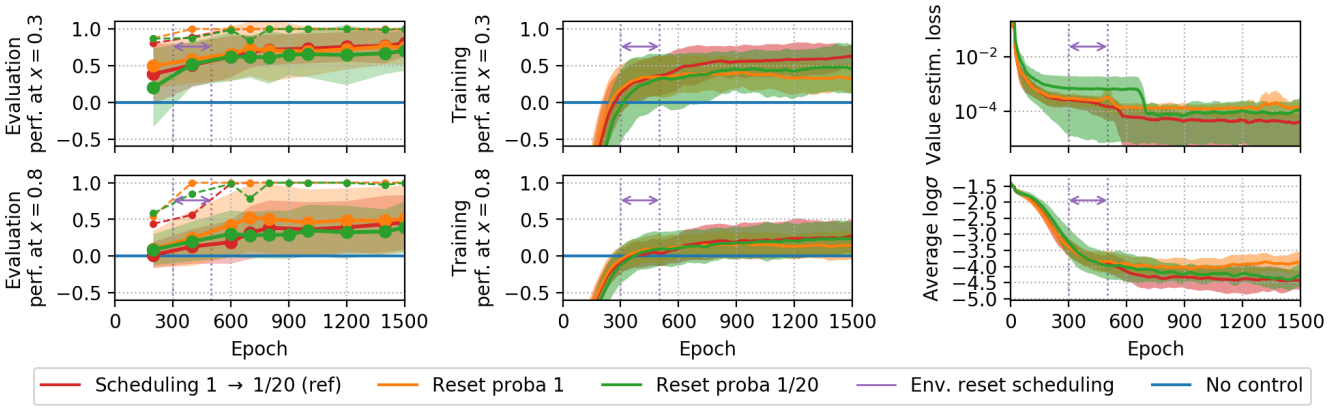


Figure 7.2: Learning curves of the last measurements of m_3 (top left and center) and m_8 (bottom left and center) metrics respectively on 4000-step evaluation runs and on training roll-outs. (top right) Evolution of the value function loss. (bottom) Evolution of the average $\log \sigma$ provided by the actor. Different scheduling setups on the environment state reset probability are compared. Solid lines represent ensemble averages (over 20-test-case batches), shaded area illustrate the corresponding standard deviation and dashed lines account for the maximum preformances.

This exemplifies the difficulty to interpret learning curves drawn for training data in order to assess the behavior of the agents in evaluation conditions. This may be especially true for noise-sensitive environments and where the control goal is to stabilize the system state. Other attempts, using longer training episodes (i.e. 750 control steps per roll-out instead of 250) and thus feeding the agent with three times more data per epoch, display slower and more expensive convergences (not shown).

A side effect of short training roll-outs, linked to value bootstrapping may even lead to policy collapse. As discussed previously, when a roll-out is artificially cut (because the prescribed number of control steps is reached), the value function estimate of the last partial state is added to the sequence of rewards in order to compute returns that are not (or minimally) impacted by this cut-off. Yet, this value function estimate may not be properly converged. And as the return values computed with it are in turn used to train the critic neural network, this creates a potentially spurious over-estimation feedback loop since the value function $V(s_t)$ is trained to match the return

R_t as:

$$V(s_t) \leftarrow R_t = \sum_{\tau=0}^{\tau_{max}} \gamma^{\tau} r_t + \gamma^{\tau_{max}} V(s_{t+\tau_{max}}).$$

The over-estimation error may propagate because of the interpolation of neural networks. For short episodes, $\gamma^{\tau_{max}}$ may on average not be small enough to damp this potential failure mode.

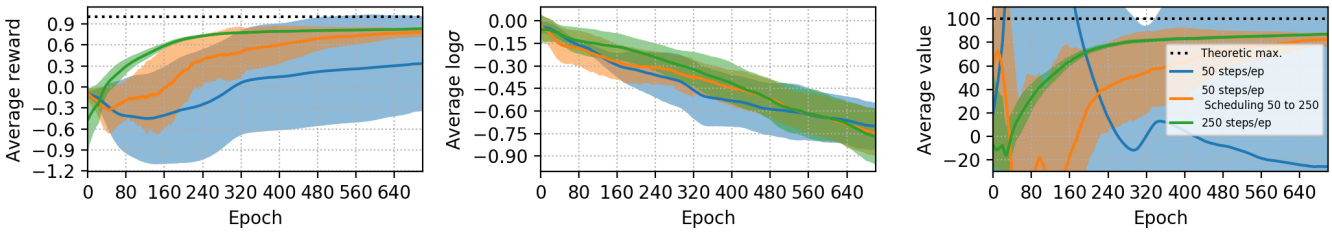


Figure 7.3: Spurious effects linked to value bootstrapping for short training roll-outs. (left) Average training reward. (center) Average $\log \sigma$. (right) Average value function estimates. Solid lines represent ensemble averages across the 10-test-case batches. Shaded areas illustrate the ensemble standard deviation.

As shown by figure 7.3, where environment reset scheduling has been tested with very short episodes, this spurious effect is detrimental to training and at least delays convergence. On short episodes (with or without further scheduling) the value function displays large standard deviations and the reward fails to improve as quickly as for longer roll-outs, where the effect of the error on the bootstrapped value are more "diluted".

7.1.2.3 Under-sampling exploration actions

Another idea would enable to still explore the state-action space efficiently without having training trajectories that end-up in chaos. Such trajectories are indeed likely to provide data samples with a low informative content with regards to the control objective. This idea resorts to only sampling an explorative control action every $n > 1$ control steps and using the best action (used for evaluation) otherwise. This enables to more easily converge value function estimators, since training data batches containing both types of control transitions (sampled and best control actions) display a much lower variance. On the other hand, this reduces by a factor $1/n$ the number of exploitable data samples for training the actor, which may represent a significant computational overhead.

Multiple preliminary tests have been launched to test the advantage of this method. So far, this enabled slightly better convergences on test-cases for which convergence is already fast without action under-sampling (such as the KS or the cylinder test-case). For larger and more complex cases such as the cavity, this method did not show significant improvements on the quality of the convergence. This may partly come from the difficulty of extrapolating evaluation-context performances from training indicators. Overall, this simple idea enabling to reduce the "signal-to-noise" ratio of the training data in an unbiased fashion regarding the stochastic estimators, calls for further investigations.

7.2 Value bootstrapping, terminal reward and solver crash

Most of the environments on which RL algorithms are developed present goals that, once reached, stop the run. For instance, video-game-like environments such as "LunarLander" from the OpenAi Gym framework stops the run once the module has landed. Conversely, robotic environments stop the run when certain joints touch the ground (indicating that the robot has fallen). Most of these cases see the terminal reward (sent upon roll-out termination) being specifically formulated. In some cases, it can even be the only one non-null.

Concerning flow control, there is generally no stopping condition when the control is successful, but an obvious failing condition that imposes a stop is a numerical solver crash. This case may happen when control imposes local conditions that, for instance, destabilize the spatial or temporal schemes or cause negative pressures or temperatures. Such conditions are detrimental to learning and should not be considered by the agent as "normal states", positive to the training. If these cases happen, the associated terminal reward should strongly penalize solver crashes in order to drive the policy away from such states. Yet, if this penalization is too strong, this may, by the generalization (interpolation) effect of NNs, distort the loss landscape so much that it may prevent from converging toward efficient policies. As explained in section 2.2.3.4, setting an *a priori* penalizing terminal reward requires to know an estimate of the average return, since, in the case of a terminal reward, no value bootstrapping is performed. Thus if the terminal reward is larger than bootstrapped values, the computed advantages for the states leading to the crash will indicate that a crash should be favored over another strategy avoiding the solver to fail. These reasons make terminal reward configuration rather tricky.

Thus, it has been chosen in the current study, to avoid solver crashes at all costs. To this end, environments are stress-tested with a series of different noisy, large-amplitude actions sequences in order to assess the maximum forcing amplitude they can withstand. In addition to action interpolation from one step to the other, this ensures these environments not to crash, but may not ensure that what happens in the vicinity of the actuators is faithful to physics 100% of the time.

Further developments should embrace a more subtle approach to the issue and consider tolerating solver crash in order to allow for more state-action exploration while still considering this event as detrimental to the training.

7.3 Input normalization: a double-edged sword

Input normalization in the sense of batch normalization (BN) is a commonly used practice consisting in re-scaling and de-biasing inputs so that they have a null mean and a unitary variance. This pre-processing generally takes the form of a additional layer upstream neural networks or layers. While showing evident benefits on the training speed and stability of the convergence, there is currently no wide consensus concerning the theoretical understanding of this efficiency.

While for supervised learning, BN simply consists in normalizing inputs across the training mini-batches (thus without requiring any additional model parameter), this cannot be the case for RL, since neural structures are queried one sample at a time (equivalently to a unitary batch size). A solution to this issue consists in storing the mean μ_{input} and standard deviation values σ_{input} used to normalize each data sample. These values still must be computed on a data batch representative of the whole input data. To do so a Polyak averaging of both mean and standard deviation of the

input is proposed:

$$\mu_{input} \leftarrow \alpha\mu_{input} + (1-\alpha)\overline{x_{inputs}},$$
$$\sigma_{input} \leftarrow \alpha\sigma_{input} + (1-\alpha)S\left(x_{input}\right),$$
$$x_{norm} = \frac{x - \mu_{input}}{\sigma_{input}},$$

where $\alpha \in [0,1]$ is a Polyak update coefficient, $\overline{x_{input}}$ is the batch-averaged input value, $S(x_{input})$ is the standard deviation across the batch, x an input and x_{norm} its corresponding normalized value. The current section explores the effects of such a mechanism on training performances.

It has been empirically observed that some cases that did not take advantage of BN had their control action saturated (clipped to either 1 or -1) and thus could not provide relevant gradients to drive training. Thus, using generally a Xavier [80] initialization, having large amplitude inputs may more likely lead to zeroed-out gradients and saturated control actions. Figure 7.4 illustrates the performances of cases trained with different BN configurations.

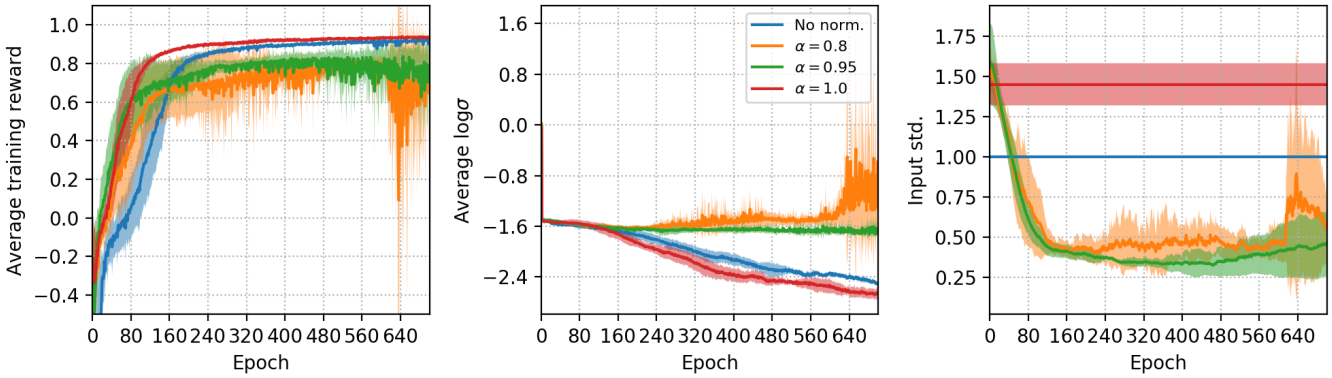


Figure 7.4: Effect of input normalization on the performances of control on the KS equation with the standard layout described in section 3.4.1.1.

On this specific case, input observations generally range $[-3,3]$ (these can be of much greater amplitude on other tests cases), thus and as shown by the non-normalized test-cases performance, input normalization is not absolutely required. However, by comparing these non-normalized cases with ones having an α value of 1 (meaning that both μ_{input} and σ_{input} are initialized on the first batch but never updated), one can notice a slightly better performance of the second especially at the beginning of training.

For α values strictly smaller than 1, on the other hand, performances are affected by the updates of the BN layer parameters. As shown by the right-hand side graph, σ_{input} decreases quickly but fails to stabilize. Having such low values and because it divides the inputs, any change in σ_{input} affects the statistics of the normalized inputs. This in turn affects the control actions, which by response of the environment return observations with different statistics. This feedback thus prevents a stabilization of σ_{input} and is then detrimental to performance. A shown by the central graph, the average exploration parameter σ of the PPO-CMA used here for training fails to decrease as it does in the previous cases. This is a sign for a lack of convergence of the policy. This lack of convergence also appears in the erratic evolution of the training reward.

On other test cases (not shown here) such as the cylinder, α values smaller than 1 do not appear as critical as for the KS. This may come from the fact that the average fluctuations of the

observations stabilize toward a larger value, for which Polyak averaging updates are not sufficient to trigger the unstable feedback described earlier. In conclusion, input normalization may bring a moderate advantage for training but α values smaller than 1 should generally be avoided, at the cost of having slightly non-normal input distributions.

7.4 Partial observation, aleatory uncertainty and the Markov hypothesis

Generally speaking, flow environments cannot be fully observed, because of their large number of degrees of freedom as well as potential measurement constraints discussed in section 2.1.3. Despite behaving deterministically and as explained in part 2.2.6.1, this partial observability causes some sort of incompressible aleatory uncertainty on the dynamics of the controlled flows.

On the other hand, the formulation of RL relies on a Markov Decision Process (MDP) that assumes the compliance with the Markov property as main characteristics. The question is then, given the forced partial observability, how compliant RL-based flow control is to the Markov property.

To bring elements to this issue, one can quantify how much extra information is brought about the next state transition by the addition of past observations and actions instead of simply the current observed partial state s. If the Markov property is valid, then this additional information is null. Otherwise, the Markov property is invalidated. Here the mutual information, previously introduced is used.

For a given number n of additional "steps back in time" one can define the mutual information between the next state s_{t+1} and previous states and actions $(s_t, a_t, \ldots, s_{t-n+1}, a_{t-n+1})$ as:

$$\mathcal{I}(n) = I\left(s_{t+1}, (s_t, a_t, \ldots, s_{t-n+1}, a_{t-n+1})\right)$$

Figure 7.5 displays the evolution of $\mathcal{I}(n)$ with the number of additional steps back in time and for different observation layouts during the training of the cylinder test case having one action component and from 1 to 6 observation measurements. This information is evaluated using a neural network trained to maximize the Donsker-Varadhan lower bound for the mutual information introduced in section 6.3.4:

$$I(X,Y) = \sup_{\phi \in \mathcal{M}_b(\Omega)} \left(\mathbb{E}_{P(X,Y)}\left[\phi\right] - \log\left(\mathbb{E}_{P(X)\otimes Y}\left[e^{\phi}\right]\right)\right),$$

where $\phi(X,Y)$ is the scalar output of the neural network and X is the following state s_{t+1}. Finally and as the quality of the convergence depends on the number of neural networks inputs, for a given number observations, inputs (i.e. past observations and actions concatenated together) have been padded so that all tested configurations have the same number of inputs. For instance, with 1 observation and 5 steps in the past, Y is $(s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_{t-4}, a_{t-4}, s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2})$. This padding does not bring any extra information (thus preserves the formulation) but allows for all neural network estimator inputs Y to have the same size (e.g. here max(steps in the past) $\times$ $(\mathrm{n_{obs}} + \mathrm{n_{act}}) = 8 \times (1+1) = 16$).

With these definitions, the deviation to the Markov property is simply $\mathcal{I}(\infty) - \mathcal{I}(1)$. One can observe that, in general all mutual information measures increase with the number of steps in the past, indicating that, even with a large number of observations, the Markov property is not valid. While with 1 observation the maximum mutual information is barely reached, with 8 steps in the past, one can consider that it is reached with 6 steps for 2 observations, with 4 steps for 3 and 4

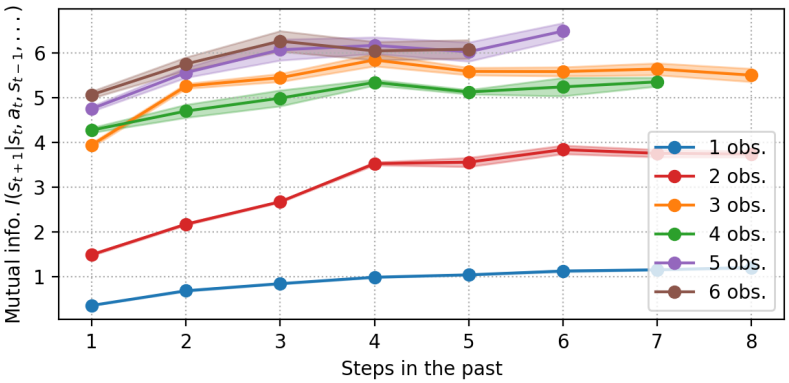


Figure 7.5: Evolution of the mutual information between s_{t+1} and past observations and actions (estimated using the Donsker-Varadhan variational formula), with respect to the number of steps in the past from which information is provided for different number of observations.

observations and finally with 3 steps for 5 and 6 observations. Thus, the more observations the faster the mutual information plateaus, indicating that more measurements in space may compensate for less measurements in time in the case of the cylinder which displays convective structures.

One can conclude that for all these tested cases the Markov property is not satisfied and yet these observation layouts enable relevant control performances on the cylinder test-case. This in turns begs the following question: Is it a serious problem not to fully comply with the Markov property underlying all the theory of RL? And if not, which degree of non-compliance can RL trainings tolerate? And what would be the appropriate measure of this deviation that would allow to preemptively know that a given observation layout is not suited for RL-based control? These open questions are worthy of further investigation since most of the real-world cases where RL-based control is considered are very likely in that same situation of partial observability causing a non-compliance to the Markov property.

7.5 Scaling-up in case complexity and cost

All test-cases studied here are relatively cheap to run compared to high-regime, high-fidelity simulations such as transitional flow DNS[2]. CFD simulation cost usually scales super-linearly with their characteristic flow non-dimensional parameters (such as y^+ wall distances or Kolomogorov-scale Reynolds numbers), requiring smaller mesh cells and smaller numerical time steps. In addition to this poor scaling, as the flow displays increasingly complex dynamics, larger observations and action layouts are required to try to control the flow. As detailed earlier, an increased number agent inputs and outputs increases its convergence time. Thus training requires more epochs ran on environments that are more expensive.

Table 7.1 reports the estimated or measured computational cost per sample and per training for different test-cases. The most expensive environments require around 10^3 to 10^4 CPUh, which already suppose convergences lasting for several days on parallelized setups. The difference of orders of magnitude with the test-cases generally used to test and validate RL algorithms already illustrates the prevalence of the sample efficiency of training algorithms over many others criteria for the feasibility of RL-based control. As with the current computing capacities, RL control is not

[2]Direct Numerical Simulations

Environment	CPU T./sample (s)	Samples/Training	CPU time (h)
Cart-Pole	3.1×10^{-4}	$10^4 \sim 10^5$	~ 0
Atari Pong	8.2×10^{-4}	10^8	~ 3
Go (AlphaZero)	10^{-4}	10^6	~ 0
2D Cylinder	1.3	10^5	~ 40
2D Cavity	8.7	5×10^5	~ 1250
2D Supersonic Jet	5.9	10^6	~ 1600
3D Hypers. Boundary Layer QDNS	28 000	10^6	~ 880 yrs.

Table 7.1: Computational cost estimations or measurements for training control laws on different test-cases. This measurements suppose a parallelized CFD, measured on a quad-core CPU Intel Xeon E5 @ 3GHz, except the last test case. – in blue: estimation. These figures do not account for the agent optimization.

conceivable on high-fidelity cases, auxiliary methods should be considered in order to circumvent these hard limitations concerning simulation costs.

Algorithms aiming at maximizing the sample efficiency to the point where only a handful of them are needed are referred to as "few-shot" learning algorithms. At the risk of over-fitting their behavior and lacking generalizability, these extract as much information from the samples as possible. Currently these methods are mostly developed in supervised or semi-supervised contexts such as computer vision (image classification or segmentation) or natural language processing (intent inference or translation) [245].

Another strategy, called "active learning", consists for the agent in selecting restarting states in order to improve in the state-space regions where it lacks performance. While being trivial to implement on simple environments, the ability to restart from a given flow state would require to have previously stored the complete snapshot corresponding to this restart. This would thus restrict restart states to already visited ones. This approach also begs the question of catastrophic forgetting discussed in section 4.1.2 and thus should be carefully crafted to avoid a complete collapse of the policy.

Policy distillation strategies detailed in section 2.2.7, belong to the larger family of transfer learning methods that consist in using the knowledge acquired by an agent in a different context. As low-regime flows share common traits with faster and/or more complex flows, this approach appears as an attractive way to save time and costs by pre-training an agent on a cheap and/or low-fidelity environment, transferring it to the target environment and only having to fine-tune its behavior with costly and time-consuming simulations. A wide spectrum of solutions ranging from simple model weight transfer to shared embedding spaces splitting both environment/agent- and task-specific kinds of knowledge [87] can be considered.

At last, this calls for innovative solutions coming from strong artificial intelligence, i.e. human experts, to help overcome these issues that are far from being solved by brute force approaches. As discussed early on, training cases already having the appropriate hyper-parameter setup may be costly, but it is often marginal compared to the computational resources required by the hyper-parameter search itself that often reveals uncertain. In hindsight from all successfully control cases, this calls for building expertise on the best practices to adopt for these searches and to share these tricks with all the stakeholders working in this area.

Chapter 8

Conclusion

The potential and challenges of Reinforcement Learning (RL) for flow control have been studied on different numerical test-cases, with the goals of:

- implementing RL control methods on flow control environment and comparing these with more traditional approaches on criteria such as the effectiveness, the energy efficiency or the robustness,
- identifying the specific challenges the application of RL to fluid mechanics stated,
- proposing solutions to these issues and testing their efficiency.

These investigations have been enabled by the development of a specific code platform interfacing a RL Application Programming Interface (API) with Computational Fluid Dynamics (CFD) solvers in a High Performance Computing (HPC) context. This modular framework, developed in Python, covers the whole training and analysis pipeline from the test-case specification to its training and post-processing and renders extensive hyper-parameter space searches easier for the end-user. It encompasses a wide range of RL and Linear Genetic Programming (LGP) agents as well as (among others) the flow control environments introduced in section 3.4.

The first results presented concerned the application of marginally adapted state-of-the-art RL and LGP training algorithms to the introduced environments. This study demonstrated the capacity of RL to deliver non-trivial control policies that are simultaneously effective, robust and energy efficient. Yet these achievements required sometimes long hyper-parameter fine-tuning and represent large computational costs. Alongside dimensionality, non-linearity and long-term dependencies, sample cost is a major identified obstacle to data-driven control design methods.

The two following chapters tackled this issue under the angle of sensor and actuator sparsity respectively. Methods that propose to reduce sensor or actuator layouts while preserving the performance as much as possible and for a reasonable computational overhead are indeed critical to the scale-up toward real-world applications. The discussed sensor sparsity algorithm simply distillates the knowledge from an expert agent using the whole observation layout to a student agent whose observations are penalized. The issue is much more complex concerning actuator sparsity. A generic action component elimination algorithm is devised and multiple ranking metrics were proposed. These were evaluated according to their performance, extra computational cost and reproducibility. Model-based was also shown to have an interesting potential on this issue by virtue of its reduced model predictive ability.

Other issues linked to sample efficiency were then discussed in chapter 7. These, not necessarily specific to flow control but rather to all scaling-up applications of RL, need to be overcome in order to enable cheaper and more reproducible results. Chaoticity and dimensionality are two major factors contributing to the emergence of these problems. Among others exploration tricks, adapted to the flow dynamics, were proposed and tested.

Numerous objectives have been fulfilled during this thesis, such as the identification of key aspects conditioning the success of RL training for flow control. Among the tackled issues, some proposed methods have provided relevant and efficient solutions while others only lay the ground work for future developments in a very young but promising field of study.

The sample cost issues has highlighted the interest of model-based approaches, an area of which the current study has barely scratched the surface. If designed and trained wisely, (non-linear) reduced order models are capable of extracting valuable parts of the underlying structure of the dynamics, similarly to their decades-old, linear counterparts but potentially having extended ranges of validity. Incorporating *a priori* knowledge via physics-informed methods or pre-constrained Partial Differential Equation (PDE) structures appears as an interesting way of learning the dynamics efficiently and thus increase sample efficiency. Model-based algorithms can indeed be thought of as multi-fidelity methods where reduced models approximating the full-state dynamics may provide relevant insights for a fraction of the cost of running the full-state, costly but accurate environment. This degree of freedom concerning the cost/quality balance of data samples appears as a valuable additional arrow in the quiver to decrease sample cost.

Expertise in fluid mechanics can also be leveraged as a means to reduce computational costs by identifying characteristic phenomena and using flow invariants to train agents on relatively cheap environments reproducing the main characteristics of more costly ones. The idea behind all transfer learning methods is indeed to pair two (or more) environments according to their similarities and the targeted control objective. As discussed previously in sections 4.4.2 and 7.5, they may be decisive for scale-up applications where for cost or safety reasons, only the fine-tuning of the agent is done on the full-state target environment and full trainings cannot be performed. Thus, developing fluid-mechanics-specific transfer learning methods, leveraging non-dimensional invariants and being capable of identifying analogous measurement locations and variables as well as actuator yielding similar effects on two different environments or implementing curriculum [71, 193] or hierarchical reinforcement [263, 164] learning is an important milestone on the path to deploying RL-based flow control to real-world applications.

Alongside sample efficiency and the need to deal with costly and sometimes ill-behaved dynamics, control robustness is a cornerstone of the future success of these methods. Thanks to the trial-and-error paradigm of RL, policies empirically benefit from a strong robustness. Yet, as climbing the Technology Readiness Levels (TRL) ladder often means passing certification processes, the issue of provable robustness will become a matter of concern. Complying with fail-safe or even safe-life design practices inevitably translates to either fitting analytic expressions on the control black-box that are neural-network-embodied policies at the risk of losing in efficiency, or to develop explainable Artificial Intelligence (AI) practices coupled with safe RL strategies. In the latter case different safety constraints may arise for both training and exploitation phases. If the training phase is performed using a simulation or in a safe environment, it does not require any specific safeguards. Otherwise one should enforce safety rules concerning the exploration of the state, to avoid dangerous ones. Such approaches are developed in the Safety-Gym API [189]. For both training and exploitation phases, Donti *et al.* [66] proposed to build provably robust neural-network policies (in the sense of a Lyapunov exponent), leveraging H_2-robust design methods (introduced

in section 2.1.6). Jeddi *et al.* [106] proposed to use Lyapunov functions to formulate training constraints ensuring robustness and also tackle the issue of out-of-distribution observations. These concerns show that, irrespective of the domain of application, the scale-up of RL is inevitably tied with robustness requirements.

In addition to promising better performances, energy efficiency or robustness, RL should foremost be seen as a formidable tool for the exploration of control solutions never thought off. Its outcomes may then question and enrich our fundamental understanding of flow dynamics. For this last single reason it is a field of study worth further investigating.

Bibliography

[1] Abadi, Martín, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey & Isard, Michael 2016 Tensorflow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* pp. 265–283.

[2] Amitay, Michael & Glezer, Ari 2002 Role of actuation frequency in controlled flow reattachment over a stalled airfoil. *AIAA Journal* **40** (2), 209–216.

[3] Andrychowicz, Marcin, Wolski, Filip, Ray, Alex, Schneider, Jonas, Fong, Rachel, Welinder, Peter, McGrew, Bob, Tobin, Josh, Abbeel, OpenAI. Pieter & Zaremba, Wojciech 2017 Hindsight experience replay. *Advances in Neural Information Processing Systems* pp. 5048–5058.

[4] Apkarian, Pierre & Noll, Dominikus 2006 Nonsmooth H_∞ synthesis. *IEEE Transactions on Automatic Control* **51** (1), 71–86.

[5] Arakeri, Jaywant H. & Shukla, Ratnesh K. 2013 A unified view of energetic efficiency in active drag reduction, thrust generation and self-propulsion through a loss coefficient with some applications. *Journal of Fluids and Structures* **41**, 22–32.

[6] Arnoult, T., Ghouila-Houri, C., Leclercq, C., Mazzamurro, A., Viard, R., Garnier, E., Sipp, D., Merlen, Alain, Talbi, A. & Pernod, P. 2021 Cavity flow controlled with an array of magneto-mechanical micro-valves. In *2021 IEEE Sensors*, pp. 1–4. IEEE.

[7] Atam, Ercan, Mathelin, Lionel & Cordier, Laurent 2016 Identification-based closed-loop control strategies for a cylinder wake flow. *IEEE Transactions on Control Systems Technology* **25** (4), 1488–1495.

[8] Bagheri, Shervin, Henningson, Dan S., Hoepffner, J. & Schmid, Peter J. 2009 Input-output analysis and control design applied to a linear model of spatially developing flows. *Applied Mechanics Reviews* **62** (2).

[9] Barbagallo, Alexandre, Sipp, Denis & Schmid, Peter 2011 Input-output measures for model reduction and closed-loop control: application to global modes. *Journal of Fluid Mechanics* **685**, 23–53.

[10] Barbagallo, Alexandre, Sipp, Denis & Schmid, Peter. J. 2009 Closed-loop control of an open cavity flow using reduced-order models. *Journal of Fluid Mechanics* **641**, 1–50.

[11] Barkley, D. 2006 Linear analysis of the cylinder wake mean flow. *EPL (Europhysics Letters)* **75** (5), 750.

[12] Barron, Andrew R. 1993 Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory* **39** (3), 930–945.

[13] Bearman, P. W. & Harvey, J. K. 1993 Control of circular cylinder flow by the use of dimples. *AIAA Journal* **31** (10), 1753–1756.

[14] Beintema, Gerben, Corbetta, Alessandro, Biferale, Luca & Toschi, Federico 2020 Controlling Rayleigh-Bénard convection via Reinforcement Learning. *Journal of Turbulence* **21** (9-10), 585–605.

[15] Belghazi, Mohamed Ishmael, Baratin, Aristide, Rajeshwar, Sai, Ozair, Sherjil, Bengio, Yoshua, Courville, Aaron & Hjelm, Devon 2018 Mutual information neural estimation. In *International conference on machine Learning*, pp. 531–540. PMLR.

[16] Bellman, Richard 1957 *Dynamic Programming*, 6th edn. Princeton University Press.

[17] Beneddine, Samir 2017 Characterization of unsteady flow behavior by linear stability analysis. PhD thesis, Université Paris-Saclay (ComUE).

[18] Beneddine, Samir, Sipp, Denis, Arnault, Anthony, Dandois, Julien & Lesshafft, Lutz 2016 Conditions for validity of mean flow stability analysis. *Journal of Fluid Mechanics* **798**, 485–504.

[19] Benoit, Christophe, Péron, Stéphanie & Landier, Sâm 2015 Cassiopee: a CFD pre-and post-processing tool. *Aerospace Science and Technology* **45**, 272–283.

[20] Bergmann, Michel & Cordier, Laurent 2008 Optimal control of the cylinder wake in the laminar regime by trust-region methods and pod reduced-order models. *Journal of Computational Physics* **227** (16), 7813–7840.

[21] Bergmann, Michel, Cordier, Laurent & Brancher, J.-P. 2005 Control of the cylinder wake in the laminar regime by trust-region methods and pod reduced order models. *Proceedings of the 44th IEEE Conference on Decision and Control* pp. 524–529.

[22] Bergmann, Michel, Cordier, Laurent & Brancher, Jean-Pierre 2006 On the generation of a reverse Von Kármán street for the controlled cylinder wake in the laminar regime. *Physics of Fluids* **18** (2), 028101.

[23] Berman, Daniel S., Buczak, Anna L., Chavis, Jeffrey S. & Corbett, Cherita L. 2019 A survey of deep learning methods for cyber security. *Information* **10** (4), 122.

[24] Bewley, Thomas R. & Liu, Sharon 1998 Optimal and robust control and estimation of linear paths to transition. *Journal of Fluid Mechanics* **365**, 305–349.

[25] Bhattacharjee, Debraj, Hemati, Maziar, Klose, Bjoern & Jacobs, Gustaaf 2018 Optimal actuator selection for airfoil separation control. *AIAA Paper* pp. 18–3692.

[26] Bieker, Katharina, Peitz, Sebastian, Brunton, Steven L., Kutz, J. Nathan & Dellnitz, Michael 2020 Deep model predictive flow control with limited sensor data and online learning. *Theoretical and computational fluid dynamics* **34** (4), 577–591.

[27] Bilanin, Alan J. & Covert, Eugene E. 1973 Estimation of possible excitation frequencies for shallow rectangular cavities. *AIAA Journal* **11** (3), 347–351.

[28] Block, Patricia J. W. 1976 Noise response of cavities of varying dimensions at subsonic speeds. *Tech. Rep.* D-8351. NASA.

[29] Botev, Zdravko I., Kroese, Dirk P., Rubinstein, Reuven Y. & L'Ecuyer, Pierre 2013 The cross-entropy method for optimization. In *Handbook of statistics*, , vol. 31, chap. 3, pp. 35–59. Elsevier.

[30] Brackston, Rowan D., De La Cruz, J. M. García, Wynn, A., Rigas, G. & Morrison, J. F. 2016 Stochastic modelling and feedback control of bistability in a turbulent bluff body wake. *Journal of Fluid Mechanics* **802**, 726–749.

[31] Braza, M., Chassaing, P. H. H. M. & Minh, H. Ha 1986 Numerical study and physical analysis of the pressure and velocity fields in the near wake of a circular cylinder. *Journal of Fluid Mechanics* **165**, 79–130.

[32] Bright, Ido, Lin, Guang & Kutz, J. Nathan 2013 Compressive sensing based machine learning strategy for characterizing the flow around a cylinder with limited pressure measurements. *Physics of Fluids* **25** (12), 127102.

[33] Brown, Tom, Mann, Benjamin, Ryder, Nick, Subbiah, Melanie, Kaplan, Jared D., Dhariwal, Prafulla, Neelakantan, Arvind, Shyam, Pranav, Sastry, Girish & Askell, Amanda 2020 Language models are few-shot learners. *Advances in neural information processing systems* **33**, 1877–1901.

[34] Bruneau, Charles-Henri & Mortazavi, Iraj 2008 Numerical modelling and passive flow control using porous media. *Computers & Fluids* **37** (5), 488–498.

[35] Brunton, Steven L. & Kutz, J. Nathan 2022 *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press.

[36] Brunton, Steven L., Proctor, Joshua L. & Kutz, J. Nathan 2016 Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences* **113** (15), 3932–3937.

[37] Brunton, Steven L., Proctor, Joshua L. & Kutz, J. Nathan 2016 Sparse identification of nonlinear dynamics with control (SINDYc). *IFAC-PapersOnLine* **49** (18), 710–715.

[38] Bucci, Michele Alessandro, Semeraro, Onofrio, Allauzen, Alexandre, Wisniewski, Guillaume, Cordier, Laurent & Mathelin, Lionel 2019 Control of chaotic systems by deep reinforcement learning. *Proceedings of the Royal Society A* **475** (2231), 20190351.

[39] Cabell, Randolph, Kegerise, Michael, Cox, David & Gibbs, Gary 2006 Experimental feedback control of flow induced cavity tones. *AIAA Journal* **44** (8), 1807–1816.

[40] Cai, Shengze, Mao, Zhiping, Wang, Zhicheng, Yin, Minglang & Karniadakis, George Em 2022 Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica* pp. 1–12.

[41] Carleo, Giuseppe, Cirac, Ignacio, Cranmer, Kyle, Daudet, Laurent, Schuld, Maria, Tishby, Naftali, Vogt-Maranto, Leslie & Zdeborová, Lenka 2019 Machine learning and the physical sciences. *Reviews of Modern Physics* **91** (4), 045002.

[42] Carpenter, Peter W. & Porter, Lee J. 2001 Effects of passive porous walls on boundary-layer instability. *AIAA Journal* **39** (4), 597–604.

[43] Cattafesta, III, L., Garg, S., Choudhari, M., Li, F., Cattafesta, III, L., Garg, S., Choudhari, M. & Li, F. 1997 *Active control of flow-induced cavity resonance*, p. 1804.

[44] Cattafesta, III, L., Shukla, D., Garg, S. & Ross, J. 1999 *Development of an adaptive weapons-bay suppression system*, p. 1901.

[45] Cattafesta III, Louis N. & Sheplak, Mark 2011 Actuators for active flow control. *Annual Review of Fluid Mechanics* **43**, 247–272.

[46] Chen, Hongming, Engkvist, Ola, Wang, Yinhai, Olivecrona, Marcus & Blaschke, Thomas 2018 The rise of deep learning in drug discovery. *Drug discovery today* **23** (6), 1241–1250.

[47] Chen, Kevin K. & Rowley, Clarence W. 2011 H 2 optimal actuator and sensor placement in the linearised complex ginzburg-landau system. *Journal of Fluid Mechanics* **681**, 241–260.

[48] Chen, Ricky T. Q., Rubanova, Yulia, Bettencourt, Jesse & Duvenaud, David K. 2018 Neural ordinary differential equations. *Advances in neural information processing systems* **31**.

[49] Chen, Zhihua & Aubry, Nadine 2005 Active control of cylinder wake. *Communications in nonlinear science and numerical simulation* **10** (2), 205–216.

[50] Chen, Zhengdao, Zhang, Jianyu, Arjovsky, Martin & Bottou, Léon 2019 Symplectic recurrent neural networks. *arXiv preprint arXiv:1909.13334* .

[51] Cho, Kyunghyun, Van Merriënboer, Bart, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger & Bengio, Yoshua 2014 Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734. Association for Computational Linguistics.

[52] Chomaz, Jean-Marc 2005 Global instabilities in spatially developing flows: non-normality and nonlinearity. *Annu. Rev. Fluid Mech.* **37**, 357–392.

[53] Chua, Kurtland, Calandra, Roberto, McAllister, Rowan & Levine, Sergey 2018 Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems* **31**.

[54] Cohen, Kelly, Siegel, Stefan & McLaughlin, Thomas 2006 A heuristic approach to effective sensor placement for modeling of a cylinder wake. *Computers & Fluids* **35** (1), 103–120.

[55] Corke, T. C. & Kusek, S. M. 1993 Resonance in axisymmetric jets with controlled helical-mode input. *Journal of Fluid Mechanics* **249**, 307–336.

[56] Cornejo Maceda, Guy Y. 2021 Gradient-enriched machine learning control exemplified for shear flows in simulations and experiments. PhD thesis.

[57] Cranmer, Miles, Sanchez-Gonzalez, Alvaro, Battaglia, Peter, Xu, Rui, Cranmer, Kyle, Spergel, David & Ho, Shirley 2020 Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems* **33**, 17429–17442.

[58] Curtiss, Charles Francis & Hirschfelder, Joseph O. 1952 Integration of stiff equations. *Proceedings of the National Academy of Sciences of the United States of America* **38** (3), 235.

[59] Cybyk, Bohdan, Grossman, Kenneth & Wilkerson, Jordan 2004 Performance characteristics of the sparkjet flow control actuator. *AIAA Paper* p. 2131.

[60] Dandois, J., Garnier, E. & Pamart, P.-Y. 2013 NARX modelling of unsteady separation control. *Experiments in fluids* **54** (2), 1445.

[61] Debien, Antoine, Von Krbek, Kai A. F. F., Mazellier, Nicolas, Duriez, Thomas, Cordier, Laurent, Noack, Bernd R., Abel, Markus W. & Kourta, Azeddine 2016 Closed-loop separation control over a sharp edge ramp using genetic programming. *Experiments in fluids* **57** (3), 40.

[62] Deisenroth, Marc & Rasmussen, Carl E. 2011 Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472.

[63] Dergham, G., Sipp, D., Robinet, J.-C. & Barbagallo, A. 2011 Model reduction for fluids using frequential snapshots. *Physics of Fluids* **23** (6), 064101.

[64] DeVries, Levi & Paley, Derek A. 2013 Observability-based optimization for flow sensing and control of an underwater vehicle in a uniform flowfield. *2013 American Control Conference* pp. 1386–1391.

[65] Donsker, Monroe D. & Varadhan, S. R. Srinivasa 1975 On a variational formula for the principal eigenvalue for operators with maximum principle. *Proceedings of the National Academy of Sciences* **72** (3), 780–783.

[66] Donti, Priya L., Roderick, Melrose, Fazlyab, Mahyar & Kolter, J. Zico 2020 Enforcing robust control guarantees within neural network policies. *arXiv preprint arXiv:2011.08105* .

[67] Doyle, John C. 1978 Guaranteed margins for LQG regulators. *IEEE Transactions on automatic Control* **23** (4), 756–757.

[68] Duriez, Thomas, Parezanovic, Vladimir, Laurentie, Jean-Charles, Fourment, Carine, Delville, Joël, Bonnet, Jean-Paul, Cordier, Laurent, Noack, Bernd R., Segond, Marc & Abel, Markus W. 2014 Closed-loop control of experimental shear flows using machine learning. *AIAA Paper* p. 2219.

[69] Edwards, Jack R. & Liou, Meng-Sing 1998 Low-diffusion flux-splitting methods for flows at all speeds. *AIAA Journal* **36** (9), 1610–1617.

[70] Evans, Humberto Bocanegra, Hamed, Ali M., Gorumlu, Serdar, Doosttalab, Ali, Aksak, Burak, Chamorro, Leonardo P. & Castillo, Luciano 2018 Engineered bio-inspired coating for passive flow control. *Proceedings of the National Academy of Sciences* **115** (6), 1210–1214.

[71] Fang, Meng, Zhou, Tianyi, Du, Yali, Han, Lei & Zhang, Zhengyou 2019 Curriculum-guided hindsight experience replay pp. 12623–12634.

[72] Foures, Dimitry P. G., Dovetta, Nicolas, Sipp, Denis & Schmid, Peter J. 2014 A data-assimilation method for Reynolds-averaged Navier-Stokes-driven mean flow reconstruction. *Journal of Fluid Mechanics* **759**, 404–431.

[73] Fujimoto, Scott, Hoof, Herke & Meger, David 2018 Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR.

[74] Gal, Yarin, McAllister, Rowan & Rasmussen, Carl Edward 2016 Improving PILCO with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, , vol. 4, p. 25.

[75] Gao, Chuanqiang, Zhang, Weiwei, Kou, Jiaqing, Liu, Yilang & Ye, Zhengyin 2017 Active control of transonic buffet flow. *Journal of Fluid Mechanics* **824**, 312.

[76] Gerhard, Johannes, Pastoor, Mark, King, Rudibert, Noack, Bernd, Dillmann, Andreas, Morzynski, Marek & Tadmor, Gilead 2003 Model-based control of vortex shedding using low-dimensional galerkin models. *AIAA Paper* p. 4262.

[77] Gers, Felix A., Schmidhuber, Jürgen & Cummins, Fred 2000 Learning to forget: Continual prediction with lstm. *Neural computation* **12** (10), 2451–2471.

[78] Giannetti, Flavio & Luchini, Paolo 2007 Structural sensitivity of the first instability of the cylinder wake. *Journal of Fluid Mechanics* **581**, 167–197.

[79] Glezer, Ari & Amitay, Michael 2002 Synthetic jets. *Annual review of fluid mechanics* **34** (1), 503–529.

[80] Glorot, Xavier & Bengio, Yoshua 2010 Understanding the difficulty of training deep feedforward neural networks. pp. 249–256. JMLR Workshop and Conference Proceedings.

[81] Greensmith, Evan, Bartlett, Peter L. & Baxter, Jonathan 2004 Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research* **5** (9).

[82] Greydanus, Samuel, Dzamba, Misko & Yosinski, Jason 2019 Hamiltonian neural networks. *Advances in neural information processing systems* **32**.

[83] Gu, Shixiang, Lillicrap, Timothy, Sutskever, Ilya & Levine, Sergey 2016 Continuous deep Q-learning with model-based acceleration. pp. 2829–2838.

[84] Gueniat, Florimond, Mathelin, Lionel & Hussaini, M. Yousuff 2016 A statistical learning strategy for closed-loop control of fluid flows. *Theoretical and Computational Fluid Dynamics* **30** (6), 497–510.

[85] Gugercin, Serkan, Antoulas, Athanasios C. & Beattie, Christopher 2008 H_2 model reduction for large-scale linear dynamical systems. *SIAM journal on matrix analysis and applications* **30** (2), 609–638.

[86] Guo, Guangming, Jiang, Sitan, Chen, Hao & Zhu, Lin 2022 Influence of flow control on aerodynamic properties of an open cavity in rarefied hypersonic flows. *Acta Astronautica* **191**, 404–416.

[87] Gupta, Abhishek, Devin, Coline, Liu, YuXuan, Abbeel, Pieter & Levine, Sergey 2017 Learning invariant feature spaces to transfer skills with reinforcement learning. In *International Conference on Learning Representations*.

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

[89] Hämäläinen, Perttu, Babadi, Amin, Ma, Xiaoxiao & Lehtinen, Jaakko 2020 PPO-CMA: Proximal policy optimization with covariance matrix adaptation. In *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6. IEEE.

[90] He, J.-W., Glowinski, R., Metcalfe, R., Nordlander, A. & Periaux, J. 2000 Active control and drag optimization for flow past a circular cylinder: I. oscillatory cylinder rotation. *Journal of Computational Physics* **163** (1), 83–117.

[91] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing & Sun, Jian 2015 Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

[92] Heller, H. & Bliss, D. 1975 *The physical mechanism of flow-induced pressure fluctuations in cavities and concepts for their suppression*, p. 491.

[93] Henderson, Ronald D. 1997 Nonlinear dynamics and pattern formation in turbulent wake transition. *Journal of Fluid Mechanics* **352**, 65–112.

[94] Henning, Lars, Pastoor, Mark, King, Rudibert, Noack, Bernd R. & Tadmor, Gilead 2007 Feedback control applied to the bluff body wake. , vol. 95, pp. 369–390. Springer.

[95] Hervé, Aurelien, Sipp, Denis, Schmid, Peter J. & Samuelides, Manuel 2012 A physics-based approach to flow control using system identification. *Journal of Fluid Mechanics* **702**, 26–58.

[96] Hinton, Geoffrey E., Sabour, Sara & Frosst, Nicholas 2018 Matrix capsules with EM routing. In *International conference on learning representations*.

[97] Hjelm, R. Devon, Fedorov, Alex, Lavoie-Marchildon, Samuel, Grewal, Karan, Bachman, Phil, Trischler, Adam & Bengio, Yoshua 2019 Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*.

[98] Hochreiter, Sepp, Bengio, Yoshua, Frasconi, Paolo & Schmidhuber, Jürgen 2001 *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. A field guide to dynamical recurrent neural networks. IEEE Press In.

[99] Hochreiter, Sepp & Schmidhuber, Jürgen 1997 Long short-term memory. *Neural computation* **9** (8), 1735–1780.

[100] Holland, John H. 1992 *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[101] Huang, Shao-Ching & Kim, John 2008 Control and system identification of a separated flow. *Physics of Fluids* **20** (10), 101509.

[102] Illingworth, Simon J., Morgans, Aimee S. & Rowley, Clarence W. 2012 Feedback control of cavity flow oscillations using simple linear models. *Journal of Fluid Mechanics* **709**, 223–248.

[103] Illy, Hervé, Geffroy, Philippe & Jacquin, Laurent 2008 Observations on the passive control of flow oscillations over a cavity in a transonic regime by means of a spanwise cylinder. *AIAA Paper* p. 3774.

[104] Ivakhnenko, Alexey Grigorevich 1971 Polynomial theory of complex systems. *IEEE transactions on Systems, Man, and Cybernetics* (4), 364–378.

[105] Janner, Michael, Fu, Justin, Zhang, Marvin & Levine, Sergey 2019 When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems* **32**.

[106] Jeddi, Ashkan B., Dehghani, Nariman L. & Shafieezadeh, Abdollah 2021 Lyapunov-based uncertainty-aware safe reinforcement learning. *arXiv preprint arXiv:2107.13944* .

[107] Jin, Bo, Illingworth, Simon J. & Sandberg, Richard D. 2020 Feedback control of vortex shedding using a resolvent-based modelling approach. *Journal of Fluid Mechanics* **897**.

[108] Jin, Bo, Illingworth, Simon J. & Sandberg, Richard D. 2022 Optimal sensor and actuator placement for feedback control of vortex shedding. *Journal of Fluid Mechanics* **932**, A2.

[109] Jin, B., Sandberg, R. D. & Illingworth, S. J. 2018 Resolvent-based feedback control of vortex shedding at low reynolds numbers. In *21st Australian Fluid Mechanics Conference (Adelaide, Australia)*, pp. 1–4.

[110] Jones, Bryn Ll, Heins, Peter H., Kerrigan, Eric C., Morrison, Jonathan F. & Sharma, Ati S. 2015 Modelling for robust feedback control of fluid flows. *Journal of Fluid Mechanics* **769**, 687–722.

[111] Joubert, Gilles, Le Pape, Arnaud, Heine, Benjamin & Huberson, Serge 2013 Vortical interactions behind deployable vortex generator for airfoil static stall control. *AIAA Journal* **51** (1), 240–252.

[112] Juang, Jer-Nan & Pappa, Richard S. 1985 An eigensystem realization algorithm for modal parameter identification and model reduction. *Journal of guidance, control, and dynamics* **8** (5), 620–627.

[113] Juang, Jer-Nan, Phan, Minh, Horta, Lucas G. & Longman, Richard W. 1993 Identification of observer/Kalman filter markov parameters - theory and experiments. *Journal of Guidance, Control, and Dynamics* **16** (2), 320–329.

[114] Jussiau, W., Leclercq, C., Demourant, F. & Apkarian, P. 2022 Learning linear feedback controllers for suppressing the vortex-shedding flow past a cylinder. *IEEE Control Systems Letters* **6**, 3212–3217.

[115] Kaelbling, Leslie Pack, Littman, Michael L. & Moore, Andrew W. 1996 Reinforcement learning: A survey. *Journal of artificial intelligence research* **4**, 237–285.

[116] Kaiser, Eurika, Kutz, J. Nathan & Brunton, Steven L. 2018 Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A* **474** (2219), 20180335.

[117] Kaiser, Lukasz, Babaeizadeh, Mohammad, Milos, Piotr, Osinski, Blazej, Campbell, Roy H., Czechowski, Konrad, Erhan, Dumitru, Finn, Chelsea, Kozakowski, Piotr & Levine, Sergey 2019 Model based reinforcement learning for atari. In *International Conference on Learning Representations*.

[118] Kalman, Rudolf E. 1960 On the general theory of control systems. In *Proceedings First International Conference on Automatic Control, Moscow, USSR*, pp. 481–492.

[119] Kang, Sangmo, Choi, Haecheon & Lee, Sangsan 1999 Laminar flow past a rotating circular cylinder. *Physics of Fluids* **11** (11), 3312–3321.

[120] Kashima, Kenji 2016 Nonlinear model reduction by deep autoencoder of noise response data. *2016 IEEE 55th Conference on Decision and Control (CDC)* pp. 5750–5755.

[121] Katoch, Sourabh, Chauhan, Sumit Singh & Kumar, Vijay 2021 A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications* **80** (5), 8091–8126.

[122] Kim, Kihwan, Kerr, Murray, Beskok, Ali & Jayasuriya, Suhada 2006 Frequency-domain based feedback control of flow separation using synthetic jets. *2006 American Control Conference* pp. 5318–5323.

[123] Kim, Samuel, Lu, Peter Y, Mukherjee, Srijon, Gilbert, Michael, Jing, Li, Čeperić, Vladimir & Soljačić, Marin 2020 Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE transactions on neural networks and learning systems* **32** (9), 4166–4177.

[124] Kingma, Diederik P. & Ba, Jimmy 2015 Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[125] Kitaev, Nikita, Kaiser, Łukasz & Levskaya, Anselm 2019 Reformer: The efficient transformer. In *International Conference on Learning Representations*.

[126] Koizumi, Hiroshi, Tsutsumi, Seiji & Shima, Eiji 2018 Feedback control of karman vortex shedding from a cylinder using deep reinforcement learning. *AIAA Paper* p. 3691.

[127] Koopman, Bernard O. 1931 Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences of the United States of America* **17** (5), 315.

[128] Korda, Milan & Mezić, Igor 2018 Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica* **93**, 149–160.

[129] Krajnović, Siniša & Fernandes, João 2011 Numerical simulation of the flow around a simplified vehicle model with active flow control. *International Journal of Heat and Fluid Flow* **32** (1), 192–200.

[130] Kumar, Vikash, Todorov, Emanuel & Levine, Sergey 2016 Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 378–383. IEEE.

[131] Kuutti, Sampo, Bowden, Richard, Jin, Yaochu, Barber, Phil & Fallah, Saber 2020 A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems* **22** (2), 712–733.

[132] Lai, Kwei-Herng, Zha, Daochen, Li, Yuening & Hu, Xia 2020 Dual policy distillation. In *Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, pp. 3146–3152.

[133] Lai, Pengyu, Wang, Rui, Zhang, Wei & Xu, Hui 2021 Parameter optimization of open-loop control of a circular cylinder by simplified reinforcement learning. *Physics of Fluids* **33** (10), 107110.

[134] Leclerc, Eric, Sagaut, Pierre & Mohammadi, Bijan 2006 On the use of incomplete sensitivities for feedback control of laminar vortex shedding. *Computers & fluids* **35** (10), 1432–1443.

[135] Leclercq, Colin, Demourant, Fabrice, Poussot-Vassal, Charles & Sipp, Denis 2019 Linear iterative method for closed-loop control of quasiperiodic flows. *Journal of Fluid Mechanics* **868**, 26–65.

[136] Lee, Keun H., Cortelezzi, Luca, Kim, John & Speyer, Jason 2001 Application of reduced-order controller to turbulent flows for drag reduction. *Physics of Fluids* **13** (5), 1321–1330.

[137] Levine, Sergey & Koltun, Vladlen 2013 Guided policy search. In *International conference on machine learning*, pp. 1–9. PMLR.

[138] Li, Hao, Maceda, Guy Y. Cornejo, Li, Yiqing, Tan, Jianguo, Morzyński, Marek & Noack, Bernd R. 2020 Towards human-interpretable, automated learning of feedback control for the mixing layer. *arXiv preprint arXiv:2008.12924* .

[139] Li, Hao, Tan, Jianguo, Gao, Zhengwang & Noack, Bernd R. 2020 Machine learning open-loop control of a mixing layer. *Physics of Fluids* **32** (11), 111701.

[140] Li, Jichao & Zhang, Mengqi 2022 Reinforcement-learning-based control of confined cylinder wakes with stability analyses. *Journal of Fluid Mechanics* **932**, A44.

[141] Li, Ruiying, Noack, Bernd R., Cordier, Laurent, Borée, Jacques, Kaiser, Eurika & Harambat, Fabien 2017 Linear genetic programming control for strongly nonlinear dynamics with frequency crosstalk. *arXiv preprint arXiv:1705.00367* .

[142] Li, Ying, Zhang, Haokui, Xue, Xizhe, Jiang, Yenan & Shen, Qiang 2018 Deep learning for remote sensing image classification: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **8** (6), e1264.

[143] Lillicrap, Timothy P., Hunt, Jonathan J., Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David & Wierstra, Daan 2016 Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*.

[144] Liu, Qiong & Gómez, F. 2019 Role of trailing-edge geometry in open cavity flow control. *AIAA Journal* **57** (2), 876–878.

[145] Louizos, Christos, Welling, Max & Kingma, Diederik P. 2018 Learning sparse neural networks through l_0 regularization. *Sixth International Conference on Learning Representations, Vancouver Canada, Monday April 30-Thursday May 03, 2018* .

[146] Luhar, Mitul, Sharma, Ati S. & McKeon, Beverley J. 2014 Opposition control within the resolvent analysis framework. *Journal of Fluid Mechanics* **749**, 597–626.

[147] Luo, Yuping, Xu, Huazhe, Li, Yuanzhi, Tian, Yuandong, Darrell, Trevor & Ma, Tengyu 2018 Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858* .

[148] Maceda, Guy Y. Cornejo, Li, Yiqing, Lusseyran, François, Morzyński, Marek & Noack, Bernd R. 2021 Stabilization of the fluidic pinball with gradient-enriched machine learning control. *Journal of Fluid Mechanics* **917**.

[149] Maceda, Guy Y. Cornejo, Varon, Eliott, Lusseyran, François & Noack, Bernd R. 2022 Stabilization of a multi-frequency open cavity flow with gradient-enriched machine learning control. *arXiv preprint arXiv:2202.01686* .

[150] Manohar, Krithika, Kutz, J Nathan & Brunton, Steven L 2021 Optimal sensor and actuator selection using balanced model reduction. *IEEE Transactions on Automatic Control* **67** (4), 2108–2115.

[151] Mao, Yiqian, Zhong, Shan & Yin, Hujun 2022 Active flow control using deep reinforcement learning with time delays in markov decision process and autoregressive policy. *Physics of Fluids* **34** (5), 053602.

[152] Marquet, Olivier, Sipp, Denis & Jacquin, Laurent 2008 Sensitivity analysis and passive control of cylinder flow. *Journal of Fluid Mechanics* **615**, 221–252.

[153] Mary, Ivan 1999 Méthode de Newton approchée pour le calcul d'écoulements instationnaires comportant des zones à très faibles nombres de Mach. PhD thesis, Paris 11.

[154] Mettot, Clément, Renac, Florent & Sipp, Denis 2014 Computation of eigenvalue sensitivity to base flow modifications in a discrete framework: Application to open-loop control. *Journal of Computational Physics* **269**, 234–258.

[155] Min, Chulhong & Choi, Haecheon 1999 Suboptimal feedback control of vortex shedding at low reynolds numbers. *Journal of Fluid Mechanics* **401**, 123–156.

[156] Mittal, Rajat & Iaccarino, Gianluca 2005 Immersed boundary methods. *Annu. Rev. Fluid Mech.* **37**, 239–261.

[157] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K. & Ostrovski, Georg 2015 Human-level control through deep reinforcement learning. *Nature* **518** (7540), 529.

[158] Moerland, Thomas M., Broekens, Joost & Jonker, Catholijn M. 2020 Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712* .

[159] Mons, Vincent, Chassaing, J.-C., Gomez, Thomas & Sagaut, Pierre 2016 Reconstruction of unsteady viscous flows using data assimilation schemes. *Journal of Computational Physics* **316**, 255–280.

[160] Mons, Vincent, Chassaing, Jean-Camille & Sagaut, Pierre 2017 Optimal sensor placement for variational data assimilation of unsteady flows past a rotationally oscillating cylinder. *Journal of Fluid Mechanics* **823**, 230–277.

[161] Mons, Vincent & Marquet, Olivier 2021 Linear and nonlinear sensor placement strategies for mean-flow reconstruction via data assimilation. *Journal of Fluid Mechanics* **923**.

[162] Moore, Bruce 1981 Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE transactions on automatic control* **26** (1), 17–32.

[163] Muddada, Sridhar & Patnaik, B. S. V. 2010 An active flow control strategy for the suppression of vortex structures behind a circular cylinder. *European Journal of Mechanics-B/Fluids* **29** (2), 93–104.

[164] Nachum, Ofir, Gu, Shixiang, Lee, Honglak & Levine, Sergey 2018 Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems* **31**.

[165] Nair, Aditya G., Taira, Kunihiko, Brunton, Bingni W. & Brunton, Steven L. 2021 Phase-based control of periodic fluid flows. *Journal of Fluid Mechanics* **927**.

[166] Natarajan, Mahesh, Freund, Jonathan B. & Bodony, Daniel J. 2016 Actuator selection and placement for localized feedback flow control. *Journal of Fluid Mechanics* **809**, 775–792.

[167] Nguyen, Giang, Dlugolinsky, Stefan, Bobák, Martin, Tran, Viet, López García, Álvaro, Heredia, Ignacio, Malík, Peter & Hluchý, Ladislav 2019 Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review* **52** (1), 77–124.

[168] Nibourel, Pierre, Leclercq, Colin, Demourant, Fabrice, Garnier, Eric & Sipp, Denis 2021 Robust control of convective instabilities in a 2d supersonic boundary layer using a feedback setup. In *AERO 2020+1 - 55th 3AF International Conference on Applied Conference*.

[169] Nishioka, Michio & Sato, Hiroshi 1978 Mechanism of determination of the shedding frequency of vortices behind a cylinder at low reynolds numbers. *Journal of Fluid Mechanics* **89** (1), 49–60.

[170] Oehler, Stephan F. & Illingworth, Simon J. 2018 Sensor and actuator placement trade-offs for a linear model of spatially developing flows. *Journal of Fluid Mechanics* **854**, 34–55.

[171] Otter, Daniel W., Medina, Julian R. & Kalita, Jugal K. 2020 A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems* **32** (2), 604–624.

[172] Ozbayoglu, Ahmet Murat, Gudelek, Mehmet Ugur & Sezer, Omer Berat 2020 Deep learning for financial applications: A survey. *Applied Soft Computing* **93**, 106384.

[173] Panju, Maysum & Ghodsi, Ali 2020 A neuro-symbolic method for solving differential and functional equations. *arXiv preprint arXiv:2011.02415* .

[174] Paris, Romain, Beneddine, Samir & Dandois, Julien 2021 Robust flow control and optimal sensor placement using deep reinforcement learning. *Journal of Fluid Mechanics* **913**, A25.

[175] Paris, Romain, Beneddine, Samir & Dandois, Julien 2022 Reinforcement-learning-based actuator selection method for active flow control. *ArXiv preprint arXiv:2209.14895* .

[176] Park, Hyungmin, Lee, Dongkon, Jeon, Woo-Pyung, Hahn, Seonghyeon, Kim, Jeonglae, Kim, Jungwoo, Choi, J. I. N. & Choi, Haecheon 2006 Drag reduction in flow over a two-dimensional bluff body with a blunt trailing edge using a new passive device. *Journal of Fluid Mechanics* **563**, 389–414.

[177] Parkin, Derwin J., Thompson, Mark Christopher & Sheridan, John 2014 Numerical analysis of bluff body wakes under periodic open-loop control. *Journal of Fluid Mechanics* **739**, 94–123.

[178] Patterson, Ryan P. & Friedmann, Peretz P. 2022 Vibration reduction on helicopter rotors using open-loop flow control. *AIAA Journal* **60** (1), 113–128.

[179] Ponsioen, Sten, Jain, Shobhit & Haller, George 2020 Model reduction to spectral submanifolds and forced-response calculation in high-dimensional mechanical systems. *Journal of Sound and Vibration* p. 115640.

[180] Post, Martiqua L. & Corke, Thomas C. 2006 Separation control using plasma actuators: dynamic stall vortex control on oscillating airfoil. *AIAA Journal* **44** (12), 3125–3135.

[181] Poussot-Vassal, C. & Sipp, D. 2015 Parametric reduced order dynamical model construction of a fluid flow control problem. *IFAC-PapersOnLine* **48** (26), 133–138.

[182] Proctor, Joshua L., Brunton, Steven L. & Kutz, J. Nathan 2016 Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems* **15** (1), 142–161.

[183] Protas, B. & Styczek, A. 2002 Optimal rotary control of the cylinder wake in the laminar regime. *Physics of Fluids* **14** (7), 2073–2087.

[184] Protas, B. & Wesfreid, J. E. 2002 Drag force in the open-loop control of the cylinder wake in the laminar regime. *Physics of Fluids* **14** (2), 810–826.

[185] Rabault, Jean, Kuchta, Miroslav, Jensen, Atle, Réglade, Ulysse & Cerardi, Nicolas 2019 Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics* **865**, 281–302.

[186] Rabault, Jean & Kuhnle, Alexander 2019 Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. *Physics of Fluids* **31** (9), 094105.

[187] Raibaudo, C. & Martinuzzi, R. J. 2021 Unsteady actuation and feedback control of the experimental fluidic pinball using genetic programming. *Experiments in Fluids* **62** (10), 1–18.

[188] Rashidi, Saman, Hayatdavoodi, Masoud & Esfahani, Javad Abolfazli 2016 Vortex shedding suppression and wake control: A review. *Ocean Engineering* **126**, 57–80.

[189] Ray, Alex, Achiam, Joshua & Amodei, Dario 2019 Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708* .

[190] Ren, Feng, Rabault, Jean & Tang, Hui 2021 Applying deep reinforcement learning to active flow control in weakly turbulent conditions. *Physics of Fluids* **33** (3), 037121.

[191] Rizi, Mohamed-Yazid, Pastur, Luc, Abbas-Turki, Mohamed, Fraigneau, Yann & Abou-Kandil, Hisham 2015 Closed-loop analysis and control of cavity shear layer oscillations. *Intl J. Flow Control* **6**, 171–187.

[192] Rogers, James 2000 A parallel approach to optimum actuator selection with a genetic algorithm. *AIAA Paper* pp. 2000–4484.

[193] Romac, Clément, Portelas, Rémy, Hofmann, Katja & Oudeyer, Pierre-Yves 2021 Teachmyagent: a benchmark for automatic curriculum learning in deep rl. In *International Conference on Machine Learning*, pp. 9052–9063. PMLR.

[194] Rosenblatt, Frank 1958 The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* **65** (6), 386.

[195] Roshko, Anatol 1953 On the development of turbulent wakes from vortex streets, report 1191. *Tech. Rep.* 2913. California Institue of Technology.

[196] Rossiter, J. E. 1964 Wind tunnel experiments on the flow over rectangular cavities at subsonic and transonic speeds. *RAE Technical Report No. 64037* .

[197] Rowley, Clarence W. 2005 Model reduction for fluids, using balanced proper orthogonal decomposition. *International Journal of Bifurcation and Chaos* **15** (03), 997–1013.

[198] Rowley, Clarence W. & Williams, David R. 2006 Dynamics and control of high-reynolds-number flow over open cavities. *Annu. Rev. Fluid Mech.* **38**, 251–276.

[199] Rowley, Clarence W., Williams, David R., Colonius, Tim, Murray, Richard M. & Macmynowski, Douglas G. 2006 Linear models for control of cavity flow oscillations. *Journal of Fluid Mechanics* **547**, 317–330.

[200] Rusu, Andrei A., Colmenarejo, Sergio Gomez, Gulcehre, Caglar, Desjardins, Guillaume, Kirkpatrick, James, Pascanu, Razvan, Mnih, Volodymyr, Kavukcuoglu, Koray & Hadsell, Raia 2015 Policy distillation. *arXiv preprint arXiv:1511.06295* .

[201] Sabour, Sara, Frosst, Nicholas & Hinton, Geoffrey E. 2017 Dynamic routing between capsules. *Advances in neural information processing systems* pp. 3856–3866.

[202] Saddington, Alistair J., Thangamani, Varun & Knowles, Kevin 2016 Comparison of passive flow control methods for a cavity in transonic flow. *Journal of Aircraft* **53** (5), 1439–1447.

[203] Samimy, M., Adamovich, I., Webb, B., Kastner, J., Hileman, J., Keshav, S. & Palm, P. 2004 Development and characterization of plasma actuators for high-speed jet control. *Experiments in Fluids* **37** (4), 577–588.

[204] Samimy, M., Debiasi, M., Caraballo, E., Serrani, A., Yuan, X., Little, J. & Myatt, J. H. 2007 Reduced-order model-based feedback control of subsonic cavity flows–an experimental approach. pp. 211–229. Springer.

[205] Sarohia, V. & Massier, P. F. 1977 Control of cavity noise. *Journal of Aircraft* **14** (9), 833–837.

[206] Sashittal, Palash & Bodony, Daniel J. 2021 Data-driven sensor placement for fluid flows. *Theoretical and Computational Fluid Dynamics* **35** (5), 709–729.

[207] Schmid, Peter J. & Sipp, Denis 2016 Linear control of oscillator and amplifier flows. *Physical Review Fluids* **1** (4), 040501.

[208] Schmidhuber, Jürgen 1992 Learning complex, extended sequences using the principle of history compression. *Neural Computation* **4** (2), 234–242.

[209] Schrittwieser, Julian, Antonoglou, Ioannis, Hubert, Thomas, Simonyan, Karen, Sifre, Laurent, Schmitt, Simon, Guez, Arthur, Lockhart, Edward, Hassabis, Demis & Graepel, Thore 2020 Mastering atari, go, chess and shogi by planning with a learned model. *Nature* **588** (7839), 604–609.

[210] Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael & Moritz, Philipp 2015 Trust region policy optimization. *International conference on machine learning* pp. 1889–1897.

[211] Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael & Abbeel, Pieter 2015 High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* .

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

[213] Seidel, Jürgen, Fagley, Casey & McLaughlin, Thomas 2018 Feedback flow control: A heuristic approach. *AIAA Journal* **56** (10), 3825–3834.

[214] Seidel, Jürgen, Siegel, Stefan, Fagley, C., Cohen, K. & McLaughlin, T. 2009 Feedback control of a circular cylinder wake. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* **223** (4), 379–392.

[215] Seifert, A., Bachar, T., Koss, D., Shepshelovich, M. & Wygnanski, I. 1993 Oscillatory blowing: a tool to delay boundary-layer separation. *AIAA Journal* **31** (11), 2052–2060.

[216] Seifert, A. & Pack, L. G. 1999 Oscillatory control of separation at high reynolds numbers. *AIAA Journal* **37** (9), 1062–1071.

[217] Selby, G. V., Lin, J. C. & Howard, F. G. 1992 Control of low-speed turbulent separated flow using jet vortex generators. *Experiments in Fluids* **12** (6), 394–400.

[218] Seshagiri, Amith, Cooper, Evan & Traub, Lance W. 2009 Effects of vortex generators on an airfoil at low reynolds numbers. *Journal of Aircraft* **46** (1), 116–122.

[219] Shimomura, Satoshi, Sekimoto, Satoshi, Oyama, Akira, Fujii, Kozo & Nishida, Hiroyuki 2020 Closed-loop flow separation control using the deep Q network over airfoil. *AIAA Journal* **58** (10), 4260–4270.

[220] Siegel, Stefan, Cohen, Kelly & McLaughlin, Tom 2003 Feedback control of a circular cylinder wake in experiment and simulation. *33rd AIAA Fluid Dynamics Conference and Exhibit* p. 3569.

[221] Silva-Ortega, M. & Assi, Gustavo Roque da Silva 2017 Suppression of the vortex-induced vibration of a circular cylinder surrounded by eight rotating wake-control cylinders. *Journal of Fluids and Structures* **74**, 401–412.

[222] Singh, Ashish & Little, Jesse 2020 Parametric study of ns-dbd plasma actuators in a turbulent mixing layer. *Experiments in fluids* **61** (2), 1–16.

[223] Singh, Abhay K. & Hahn, Juergen 2005 Determining optimal sensor locations for state and parameter estimation for stable nonlinear systems. *Industrial & engineering chemistry research* **44** (15), 5645–5659.

[224] Singha, Sintu & Sinhamahapatra, K. P. 2011 Control of vortex shedding from a circular cylinder using imposed transverse magnetic field. *International Journal of Numerical Methods for Heat & Fluid Flow* **21** (1), 32–45.

[225] Sipp, Denis 2012 Open-loop control of cavity oscillations with harmonic forcings. *Journal of Fluid Mechanics* **708**, 439–468.

[226] Sipp, Denis & Lebedev, Anton 2007 Global stability of base and mean flows: a general approach and its applications to cylinder and open cavity flows. *Journal of Fluid Mechanics* **593**, 333–358.

[227] Sipp, Denis, Marquet, Olivier, Meliga, Philippe & Barbagallo, Alexandre 2010 Dynamics and control of global instabilities in open-flows: a linearized approach. *Applied Mechanics Reviews* **63** (3).

[228] Sipp, Denis & Schmid, Peter J. 2016 Linear closed-loop control of fluid instabilities and noise-induced perturbations: A review of approaches and tools. *Applied Mechanics Reviews* **68** (2).

[229] Sivashinsky, Gregory I. 1980 On flame propagation under conditions of stoichiometry. *SIAM Journal on Applied Mathematics* **39** (1), 67–82.

[230] Smith, Barton, Glezer, Ari, Smith, Barton & Glezer, Ari 1997 Vectoring and small-scale motions effected in free shear flows using synthetic jet actuators. In *35th aerospace sciences meeting and exhibit*, p. 213.

[231] Sohankar, A., Khodadadi, M. & Rangraz, E. 2015 Control of fluid flow and heat transfer around a square cylinder by uniform suction and blowing at low Reynolds numbers. *Computers & Fluids* **109**, 155–167.

[232] Srivastava, Rupesh Kumar, Greff, Klaus & Schmidhuber, Jürgen 2015 Highway networks. *arXiv preprint arXiv:1505.00387* .

[233] Strykowski, P. J. & Hannemann, K. 1991 Temporal simulation of the wake behind a circular cylinder in the neighborhood of the critical reynolds number. *Acta mechanica* **90** (1), 1–20.

[234] Strykowski, Pj J. & Sreenivasan, Kr R. 1990 On the formation and suppression of vortex 'shedding'at low reynolds numbers. *Journal of Fluid Mechanics* **218**, 71–107.

[235] Sutton, Richard S & Barto, Andrew G 1998 *Introduction to reinforcement learning*. The MIT press.

[236] Suzuki, H., Kasagi, N. & Suzuki, Y. 2004 Active control of an axisymmetric jet with distributed electromagnetic flap actuators. *Experiments in fluids* **36** (3), 498–509.

[237] Tang, Hongwei, Rabault, Jean, Kuhnle, Alexander, Wang, Yan & Wang, Tongguang 2020 Robust active flow control over a range of reynolds numbers using an artificial neural network trained through deep reinforcement learning. *Physics of Fluids* **32** (5), 053605.

[238] Teh, Yee, Bapst, Victor, Czarnecki, Wojciech M., Quan, John, Kirkpatrick, James, Hadsell, Raia, Heess, Nicolas & Pascanu, Razvan 2017 Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems* **30**.

[239] Tol, H. J., Kotsonis, M., De Visser, C. C. & Bamieh, B. 2017 Localised estimation and control of linear instabilities in two-dimensional wall-bounded shear flows. *Journal of Fluid Mechanics* **824**, 818–865.

[240] Van Hasselt, Hado, Guez, Arthur & Silver, David 2016 Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*.

[241] Verma, Siddhartha, Papadimitriou, Costas, Lüthen, Nora, Arampatzis, Georgios & Koumoutsakos, Petros 2020 Optimal sensor placement for artificial swimmers. *Journal of Fluid Mechanics* **884**.

[242] Vona, Marco & Lauga, Eric 2021 Stabilizing viscous extensional flows using reinforcement learning. *Physical Review E* **104** (5), 055108.

[243] Waldock, Antony, Greatwood, Colin, Salama, Francis & Richardson, Thomas 2018 Learning to perform a perched landing on the ground using deep reinforcement learning. *Journal of Intelligent & Robotic Systems* **92** (3-4), 685–704.

[244] Wang, Jin-Jun, Choi, Kwing-So, Feng, Li-Hao, Jukes, Timothy N. & Whalley, Richard D. 2013 Recent developments in dbd plasma flow control. *Progress in Aerospace Sciences* **62**, 52–78.

[245] Wang, Yaqing, Yao, Quanming, Kwok, James T. & Ni, Lionel M. 2020 Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* **53** (3), 1–34.

[246] Wang, Yi-Zhe, Mei, Yu-Fei, Aubry, Nadine, Chen, Zhihua, Wu, Peng & Wu, Wei-Tao 2022 Deep reinforcement learning based synthetic jet control on disturbed flow over airfoil. *Physics of Fluids* **34** (3), 033606.

[247] Wehrmann, O. H. 1965 Reduction of velocity fluctuations in a karman vortex street by a vibrating cylinder. *The Physics of Fluids* **8** (4), 760–761.

[248] Werbos, Paul 1974 Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University* .

[249] Willcox, Karen 2006 Unsteady flow sensing and estimation via the gappy proper orthogonal decomposition. *Computers & Fluids* **35** (2), 208–226.

[250] Willcox, Karen & Peraire, Jaime 2002 Balanced model reduction via the proper orthogonal decomposition. *AIAA Journal* **40** (11), 2323–2330.

[251] Williams, David & Morrow, Julie 2001 Adaptive control of multiple acoustic modes in cavities. In *15th AIAA Computational Fluid Dynamics Conference*, p. 2769.

[252] Williams, David R., Cornelius, Daniel & Rowley, Clarence W. 2007 Supersonic cavity response to open-loop forcing. pp. 230–243. Springer.

[253] Williams, Matthew O., Hemati, Maziar S., Dawson, Scott T. M., Kevrekidis, Ioannis G. & Rowley, Clarence W. 2016 Extending data-driven koopman analysis to actuated systems. *IFAC-PapersOnLine* **49** (18), 704–709.

[254] Williams, Matthew O., Kevrekidis, Ioannis G. & Rowley, Clarence W. 2015 A data-driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science* **25** (6), 1307–1346.

[255] Williamson, Charles H. K. 1996 Vortex dynamics in the cylinder wake. *Annual review of fluid mechanics* **28** (1), 477–539.

[256] Wong, H. Y. & Kokkalis, A. 1982 A comparative study of three aerodynamic devices for suppressing vortex-induced oscillation. *Journal of Wind Engineering and Industrial Aerodynamics* **10** (1), 21–29.

[257] Wu, Jie-Zhi, Lu, Xi-Yun, Denny, Andrew G., Fan, Meng & Wu, Jain-Ming 1998 Post-stall flow control on an airfoil by local unsteady forcing. *Journal of Fluid Mechanics* **371**, 21–58.

[258] Wu, J. Z., Vakili, A. D. & Wu, J. M. 1991 Review of the physics of enhancing vortex lift by unsteady excitation. *Progress in Aerospace Sciences* **28** (2), 73–131.

[259] Wygnanski, Israel 1997 Boundary layer and flow control by periodic addition of momentum. In *4th Shear Flow Control Conference*, pp. 97–2117.

[260] Yamouni, S., Mettot, C., Sipp, D. & Jacquin, L. 2013 Passive control of cavity flows. *Aerospace Lab* (6), p–1.

[261] Yao, Huaijin, Sun, Yiyang & Hemati, Maziar S. 2022 Feedback control of transitional shear flows: Sensor selection for performance recovery. *Theoretical and Computational Fluid Dynamics* **36** (4), 597–626.

[262] Yeh, Chi-An & Taira, Kunihiko 2019 Resolvent-analysis-based design of airfoil separation control. *Journal of Fluid Mechanics* **867**, 572–610.

[263] Yin, Haiyan & Pan, Sinno Jialin 2017 Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *Thirty-First AAAI conference on artificial intelligence*.

[264] Yin, Yuan, Le Guen, Vincent, Dona, Jérémie, de Bézenac, Emmanuel, Ayed, Ibrahim, Thome, Nicolas & Gallinari, Patrick 2021 Augmenting physical models with deep networks for complex dynamics forecasting. *Journal of Statistical Mechanics: Theory and Experiment* **2021** (12), 124012.

[265] Zames, George 1981 Feedback and optimal sensitivity: Model reference transformations, multiplicative seminorms, and approximate inverses. *IEEE Transactions on automatic control* **26** (2), 301–320.

[266] Zeng, Kevin & Graham, Michael D. 2021 Symmetry reduction for deep reinforcement learning active control of chaotic spatiotemporal dynamics. *Physical Review E* **104** (1), 014210.

[267] Zielinska, B. J. A., Goujon-Durand, S., Dusek, J. & Wesfreid, J. E. 1997 Strongly nonlinear effect in unstable wakes. *Physical Review Letters* **79** (20), 3893.

Chapter 9

Appendices

9.1 Miscellaneous details about the RLFramework

9.1.1 Common FastS envrionment Mixin

As detailed in section 3.2.2.2, all the environments using the FastS solver should display a similar behavior on many aspects. Thus it is logical to gather these common traits into a common toolbox, a `Mixin` centralizing the code at a single location to reduce the development burden of a new environments and increase maintainability. Alongside `fasts_utils` (containing CFD specific functions), the `FastsEnvMixin` gathers methods that enable the easy definition of the observation and action setups, the reward and other complementary information sources as well as common solver warm-up operations.

9.1.1.1 Forcing action definition

In the current version, only 2D flows are supported and forcing actions can either take the form of a wall-mounted blowing/suction actuator or of a body force implemented directly as a source term in the Navier-Stokes equations.

For the latter and as the forcing action is updated at every numerical step, source terms arrays are computed by superimposition of multiple masks, one per forcing action, that multiplied by the corresponding action component and summed provide a resultant source term for all conservative variables (density ρ, x and y momentum ρu and ρv and the energy density ρe). For topological reasons the computational domain can be split into multiple zones and as masks can span several zones, this computation theoretically occurs for all the zones of the flow. Yet as body forces are defined using a target point (x, y) in the flow on which a bi-dimensional Gaussian distribution is centered, the latter decaying quickly with the distance to this point, the area where the action mask is significant remains limited. The mask is then thresholded below a fixed value.

Injection/suction forcings are more complex since they require to directly modify the case simulated by adding a boundary condition mimicking blowing and suction. To specify such a forcing, the user defines a set of new boundary conditions that are created by splitting existing ones in the simulation tree. These nodes are initialized as "walls", corresponding to a neutral action (no forcing). Depending on the sign of the action, their type may change since blowing or suction are simulated by different solver boundary conditions. These changes need to be passed on the solver. This is done by re-running part of the warm-up process every time the type of boundary condition

changes. Otherwise, when only the amplitude of the forcing action changes, array values are simply updated.

Both injection/suction and body force control actions are defined using a standardized Python dictionary (`dict`) containing all the required information. To alleviate the configuration encoding burden, some default values can be omitted and required ones are pre-processed by methods of the `Mixin` to either convert a user-defined "light" configuration to a standardized one (mainly consisting in vectorizing scalar parameters) or the raise error messages when the provided configuration is too invalid to be standardized without performing possibly unwanted adaptations of the configuration.

Again in order to make case specification easier, the definition of new boundary conditions for injection/suction forcing can be performed by directly pointing at the target geometrical range and specifying the required splitting (number and intersperse pattern). The `Mixin` then handles the splitting and the renaming of the new boundary conditions.

For instance:

```
dict(inj_BC_name='INJ_',target_BC_name='AIRFOIL',start_xy=(0,0),stop_xy=(1,0),
    intersperse_wall_bc=1,n_split=10)
```

specifies the creation of 10 boundary condition nodes named `INJ_0` to `INJ_9` on the boundary condition named "AIRFOIL", in the area delimited by the nodes closest to `start_xy` and `stop_xy`. In-between each of these new boundary conditions, a wall boundary condition of the same length is interspersed.

For both types of control actions, the update is simply performed by calling a function that is created upon the instantiation of the environment. This function keeps track of all the handles (pointers to arrays and data buffers or default values) necessary to updating the forcing in the simulation.

9.1.1.2 The action update pipeline

When the control action a_{agent} is received by the environment at the beginning of a control step it is first pre-processed. Actions coming from the agent generally range $[-1, 1]$ whereas the environment action range may be different, one agent action component may be implemented at multiple locations on the flow (e.g. using pairs of actuators that act in opposition) or this action may specify an amplitude and a frequency of sinusoidal forcing instead of simple amplitude command. Finally the user may schedule the action of the layout by redistributing the agent actions differently on the flow.

This pre-processing is done by using a first redistribution matrix $\underline{\underline{M}}_{redist}$ (by default the identity matrix) then rescaling (using the matrix $\underline{\underline{M}}_{rescale}$ and the offset vector $\underline{v}_{offset}$) and broadcasting the agent actions from $[-1, 1]$ to the action ranges defined by the configuration. Extra action bias and penalties can be added at this stage, but this is generally omitted. The environment action a_{env} ready to be implemented on the flow is thus defined as:

$$a_{env} = \underline{\underline{M}}_{rescale}.\underline{\underline{M}}_{redist}.a_{agent} + \underline{v}_{offset}$$

and it contains re-scaled amplitudes as well as frequencies for sinusoidal forcings. Along the numerical iterations of the control step, an interpolation between from the previous action to the current one is generally performed over a fixed portion of the control step in order to prevent solver crashes due to brutal changes of conditions. For each numerical iteration of the state, the implemented action is thus interpolated. In addition, for sinusoidal forcing, the interpolation of the amplitude and frequency comes along the retrieval and update of the current phase φ of the signal to guarantee a continuous forcing signal.

9.1.1.3 Flexible measurement and reward definitions

The definition and functional structure of observations and other information measurements follow the same logic as for the control actions. Measurements on the flow can be of different types, measuring different physical quantities (velocity, pressure, vorticity) as well as having different footprints in space and time (point-wise localized measurement, measurement integrated over an area, linear or surface measurement for convolutional agents or measurement with memory for time-recurrent NN structures), and different signal filtering methods (computing statistics, finite-impulse-response filtering or continuous-wavelet transform). The `Mixin` also offers to sample measurements at rate different from the control step, meaning that over- or under-sampling can be performed in order to enable more precise filtering.

Each measurement is implemented via a purposefully built function tasked with computing these physical quantities and with appending the data sample into dedicated buffers, that enable easy querying and post-processing. Measurements that aim at being used as observations for the agent are defined by the observation setup while other informative ones may be needed for simple information or more importantly for reward computation.

The reward function is thus defined as a linear combination of terms computed on the information, observations or action buffers. Similarly as for the action setup, observations, reward and other information measurement are defined using lightweight dictionary configuration that are parsed by the `Mixin` in order to generate all the `callable`s required to run the environment properly.

For instance:

```
info_setup=[
    dict(name='press',type='pointwise',args=[(0.8,y,0) for y in np.linspace(-.1,.1,5)],
    var='Pressure',memory=20),
    dict(name='vx',type='pointwise',args=[(0.8,y,0) for y in np.linspace(-.1,.1,5)],
    var='VelocityX',memory=20)
],
rew_setup=[
        dict(type='std',var='press_*',factor=-1),
        dict(type='absAvg',var='press_*',factor=-.5),
        dict(type='std',var='vx_*',factor=-.1),
        dict(type='absAvg',var='action_*',factor=-1.)
],
```

defines two groups of measurements, one of the pressure and the other of the X velocity, at location specified by `args` and stored in buffers having 20 slots of memory. These buffers are named `press_0`, ...,`press_4` and `vx_0`, ... `vx_4`. The reward function is then defined as:

$$-1 \times \frac{1}{5}\sum_{i=0}^{4} \text{std}(\text{press\_i}) - 0.5 \times \sum_{i=0}^{4} \text{avg}(|\text{press\_i}|) - 0.1 \times \frac{1}{4}\sum_{i=0}^{4} \text{std}(\text{vx\_i}) - 0.1 \times \frac{1}{n}\sum_{i=0}^{n-1} \text{avg}(|\text{action\_i}|)$$

where `action_i` refers to the i^{th} action buffer. This way terms penalizing the pressure fluctuation and x velocity signals are combined with others that encourage pressure and forcing action with a null time-averaged value.

9.1.2 EnvMPICommunicator implementation details

As introduced in section 3.2, the `envMPIcommunicator` object acts as a switch between the agent rank and the environment ranks. To preserve the modularity as much as possible, agents (expect concerning their data buffers) are implemented in a fashion agnostic to the parallelism. As in sequential mode, both environment and agent are on the same process, the agent gets a handle on the environment enabling it to directly query the observations and the reward. Conversely the learner requests the control action from the agent and runs the environment forward using this action. In a parallel mode, the environment not being run on the same rank as the agent, the latter cannot directly access observations and reward directly. The role of the `envMPIcommunicator` is then to fake the presence of an environment on the agent rank and similarly provide the queryable methods exposed by the agent on environment ranks. Here, the importance of the interface contract appears clearly as the `envMPIcommunicator` should comply with the environment interface on the agent rank and with the agent interface on environment ranks.

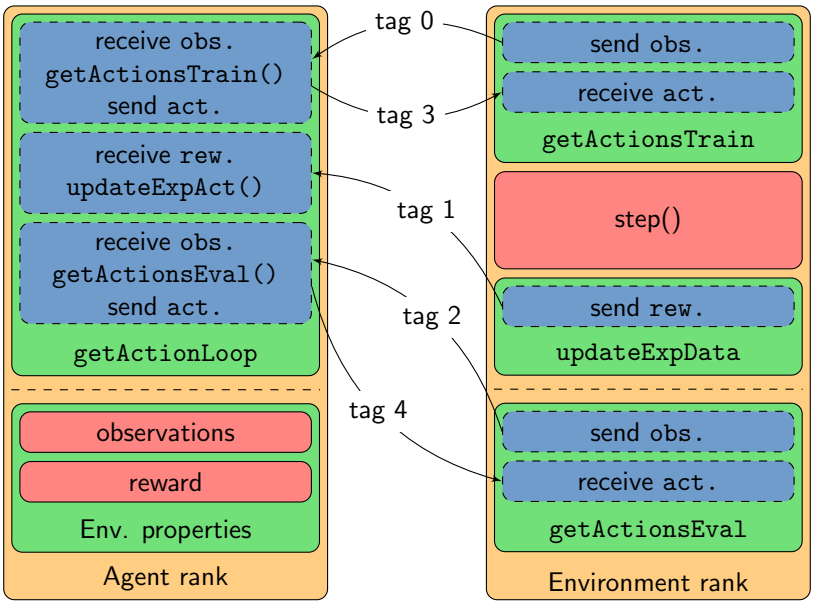


Figure 9.1: Schematics of MPI messaging managed by the `EnvMPIcommunicator` (green boxes).

Under the hood and upon one epoch of training, an "action loop" is launched on the agent rank and as illustrated by figure 9.1. Since one can deterministically compute the number of data samples exchanged in-between the agent and the environment, the agent rank is set on hold, waiting for a precise number of packets from the other ranks. As each MPI message contains the identity of its emitter and as every rank gets to compute the mapping between ranks, environments and agents, each message can be traced back to which entity sent it. Added with a tag (an integer sent alongside the message to avoid mixing simultaneous messages), used to encode the nature of the request, can also be recovered, the receiving rank gets all the information for processing the received message correctly, i.e. sending a control action back (tag 0 or 2 depending on train or evaluation contexts) with the tag 3 or storing the reward (tag 1) in the buffer corresponding to emitter environment.

At the other ends of the line, on environment ranks, the `envMPIcommunicator` exposes both the `getActionsTrain` and `getActionsEval` methods and the `updateExpData` (that normally queries the reward from the environment and pushes the triplet action, observation and reward to the right buffer), but as "empty shells" that send or receive MPI messages to the agent rank.

Other functions allowing for instance for the agent to directly modify the configuration environments are also implemented but not used on a general basis since they break the interface contract in some sense.

9.1.3 Parallelized LGP learners and agent structures

As the paradigm of LGP is different from the RL, notably in the fact that the notion of training epoch is much less well-defined in the first case, the training pipeline varies. This also enables to optimize the computational load on each rank. While still keeping the notion of an agent rank that is considered as the main one, other ranks can either run "environment" tasks as well as secondary agent optimization tasks (called background tasks). The latter can represent a significant computational overhead that has merit in being shared across all available computational resources. Thus, to enable this flexibility, a LPG learner is built around the notion of task.

The main agent rank gets a `LGPOptimizer` tasked with managing the population of policies generally. Each environment rank implements an environment and a `LGPEvaluator`, the latter being able to either run evaluation of individuals or perform background tasks. It can be seen as a worker, receiving a queue of tasks coming with the relevant data. The dispatch of all the tasks to be run at a given time is performed by a `LGPcommunicator`. These classes implement standard LGP algorithms both also a variant proposed by Cornejo Maceda [56] that infers an estimation the local gradient of the performance metric to guide the evolution of the population.

9.1.4 Multi-policy distillation

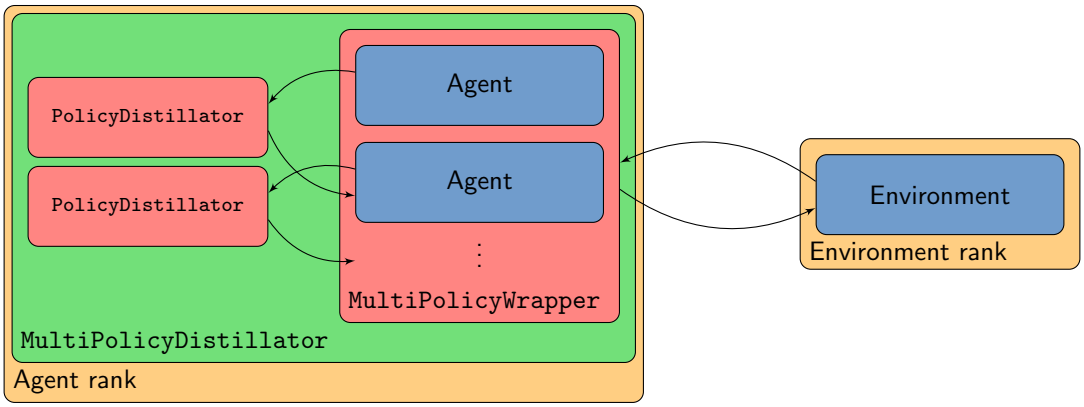


Figure 9.2: Schematics of the multi-policy distillation implemented in the RLFramework

Multi-policy distillation has been implemented alongside standard transfer learning methods in order to enable easy exchange of knowledge between RL agents and/or GP-trained individuals. As distillation considers roll-out data as a training base, (parallelized) interactions with environments, complying the environment interface must be ensured. To do so, a `MultiPolicyDistillator` has been developed with the role of a learner (high level API). In addition to the standard training of agents, it implements distillation processes between agents.

The all-to-all distillation mapping decomposed into directional, pairwise distillation relations between one teacher agent and one student agent. Each of these distillations are handled by a `PolicyDistillator` object. Depending on the type of teacher agent (either a neural-network policy or a LGP individual), the appropriate version of distillator is spawned and in turn depending

on the training algorithm, different losses, and variable handles are recovered to enable a proper optimization. These arduous and error-prone recovery operation are performed transparently to the user.

All the agents are gathered into a `MultiPolicyWrapper` object whose main task is to host multiple agents, (compliant to the agent interface) while being itself compliant to the agent interface and enable coherent training of each individual agent. It thus sequentially exposes one agent after the other for it to be trained in a scheduled manner. This wrapper thus serves essentially as switch, looking like an agent from outside objects but hosting multiple ones.

9.2 The Laplace Transform for linear closed-loop control

For a given temporal signal f, its Laplace transform F reads:

$$F(s) = \mathcal{L}\left[f(t)\right](s) = \int_0^\infty f(t)e^{-st}dt,$$

where s is a complex frequency, generally decomposed as $s = \sigma + \imath\omega$, where σ and ω respectively its real and imaginary parts. Linear closed-loop control generally considers LTI plants whose response $y(t)$ to an input signal $x(t)$ can be computed as:

$$y(t) = x(t) * h(t) = \int_0^t x(t-\tau)h(\tau)d\tau,$$

where $*$ represents temporal convolution and $h(t)$ is the impulse response of the plant. The Laplace Transform trivializes the temporal convolution as simple product in the s-domain:

$$Y(s) = H(s)X(s),$$

where X, Y and H are respectively the Laplace transforms of x, y and h, the latter being referred to as the "transfer function" of the plant. This property for instance enables to estimate H by a simple ratio of the Laplace transforms of the output over the input signals. Transfer functions can be represented in different ways, one of which being (under mild assumptions) the state space model introduced earlier in the development.

For H_2-optimal syntheses, pole placement is performed in the s-domain. Parseval's theorem ensures that if a closed-loop transfer function is H_2 optimal it is also optimal in the temporal domain. The Parseval identity reads:

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} \left|F(\sigma + \imath\omega)\right|^2 d\omega = \int_0^\infty e^{-2\sigma t}\left|f(t)\right|^2 dt.$$

The general interpretation is Parseval identity is that the Laplace preserves the energy of the signal, i.e. computing by summing the power per sample across time or frequency leads to the same result. This Parseval identity holds and for functions f that comply with the assumption: $e^{-ct}f(t) \in \mathbf{L}^2(0,\infty)$ for some $c \in \mathrm{R}$. It is then valid for $\sigma > c$, (the integration in the s-domain along a "vertical line" converges in the right half-plane defined by $\sigma > c$). The functions f that comply with this property define the Hardy space H^2.

9.3 Global stability analysis and Newton convergence method

Global stability tools developed by Samir Beneddine [17] in Python 2.7 have been brought to Python 3.x standard, re-using some building blocks developed for the RlFramework. While being directly used for RL or LGP, these tools were used for the development of some environments as a way to compute base-flows, resolvent operators or global stability analyses.

The core function of this module is the computation of a Jacobian matrix of a flow by the finite-difference method. The base idea is to perturb each degree of freedom of a steady flow $\underline{q}_0$ with a perturbation $\underline{q}'$ and to extract the residuals (as a response of the Navier-Stokes operator $\mathcal{N}(\underline{q}_0 + \underline{q})$) due to this perturbation:

$$\begin{aligned} \frac{\partial \underline{q}_0}{\partial t} &= \mathcal{N}(\underline{q}_0) = 0 \\ \frac{\partial \underline{q}'}{\partial t} &= \mathcal{N}(\underline{q}_0 + \underline{q}') \approx \underbrace{\left.\frac{\partial \mathcal{N}}{\partial \underline{q}}\right|_{\underline{q}_0}}_{\underline{\underline{J}}} .\underline{q}', \end{aligned}$$

where $\underline{\underline{J}}$ is the Jacobian operator of the flow $\underline{q}_0$ (obtained here by a first order Taylor series). Leveraging the fact that the support of the numerical stencil is limited in space, the response of the discretized Navier-Stokes operator is also limited in space. This makes it possible for two sufficiently distant perturbation to not overlap their corresponding responses of the flow. Thus using this fact, Samir Beneddine derived a strategy to cleverly "tile" perturbations with respect to the stencil in order to retrieve entire groups of rows of the Jacobian matrix at once and thus enabling the computation of Jacobian matrices by the finite-difference method on much larger flows. Originally developed for one single type of bi-dimensional stencil, for which the optimal tiling can be easily derived, this new version implements an algorithm that derives the optimal tiling for every provided stencil or retrieves this optimal solution if the stencil has already been submitted. This aims at enabling the computation of global stability with compact and/or higher order numerical schemes.

The adaptation of the module also consisted into modifying the way the different flow zone connections were handled. These zone connections define the global topology of the simulation and thus constrain the perturbation tiling previously introduced. Instead of trying to solve potential conflicts, this new version directly "disconnects" the zones by replacing each neighbor of a given zone by shallow versions ensuring the validity of the computation of the response of the flow while getting rid of the potential conflicts of patterns.

Other derived functionalities such as computation of the spectrum of the Jacobian, computation of the resolvent operator and its optimal forcings and responses or the Newton method, leveraging the Jacobian to completely stabilize a flow were also re-implemented in this new version. These functions use the PETSc and SLEPc APIs that enable massively parallel linear algebra computations.

French summary / Résumé en français

Cette thèse vise à évaluer le potentiel ainsi que les difficultés posées par l'utilisation de l'apprentissage profond par renforcement (DRL) pour le contrôle des écoulements. Le premier est un domaine de l'intelligence artificielle qui vise à optimiser le comportement d'un agent en interaction avec son environnement dans le but d'atteindre un objectif (préalablement fixé) le plus efficacement possible. Le DRL profite des récents développements notamment en Machine Learning, qui ont vu des percées significatives dans des domaines variés tels que la traduction automatique ou encore la vision par ordinateur. Le second repose principalement sur les fondamentaux de l'automatique linéaire, mais à la différence de cette dernière, il s'adresse à des systèmes (les écoulements) ayant un grand nombre de degrés de liberté (10^4 à 10^6). Les méthodes traditionellement utilisées s'attachent donc à modéliser ces systèmes linéairement (afin de profiter de la puissance du formalisme algébrique de l'automatique) tout en réduisant le nombre de degrés de liberté du modèle réduit pour rendre le problème tractable informatiquement. Cette thèse se propose de s'affranchir de cette hypothèse de linéarité en utilisant le DRL comme outil de synthèse de lois de contrôle et d'évaluer le potentiel ainsi que les défis posés par l'association de ces deux domaines.

Ces études reposent sur le développement d'une base de code ayant pour but la mise en oeuvre de larges études hyper-paramétriques en tirant parti de la puissance des super-calculateurs mis à disposition par l'ONERA. Ces modules, développés en Python visent majoritairement à permettre l'interface entre deux outils déjà optimisés. L'API Tensorflow (1.14) d'une part, implémente la gestion des réseaux de neurones ainsi que leur optimisation par auto-différenciation. D'autre part les simulations numériques des écoulements sont supportées par le code CFD FastS (développé à l'ONERA). La principale valeur ajoutée de ce développement réside dans l'ordonnancement et la gestion parallélisée des flux de données entre ces deux briques logicielles. Construit de façon modulaire, ce block de code prévoit une intégration facilitée de nouveaux environnements, agents et codes de calcul CFD. L'évaluation du potentiel ainsi que des obstacles s'opposant à la généralisation de l'usage du DRL pour le contrôle des écoulements s'est faite au travers de l'étude de différents cas tests académiques, de phénoménologie et de complexité variable. Ceux-ci reproduisent, pour un coût modéré des mécanismes dynamiques que l'ont souhaite contrôler afin d'en modifier les effets.

Les principaux cas et écoulements étudiées sont d'abord l'équation mono-dimensionnelle de Kuramoto-Sivashinsky dont la dynamique quasi-chaotique est analogue à la propagation des frontsde flamme dans les mélanges de gaz carburant-comburant. Ce cas très peu coûteux en calcul permet notamment une première comparaison entre les algorithmes qualifiés de model-free (où l'agent d'apprentissage ne modélise par la dynamique entrée-sortie de l'environnement) et ceux dits model-based (où un modèle réduit dynamique est appris). Les problématiques d'oubli catastrophique sont également explorées par la formulation de multiples objectifs de contrôle pour un même agent. L'étude du contrôle d'une allée tourbillonnaire de Van Karman dans le sillage d'un cylindre à bas nombre de Reynolds permet ensuite de comparer la performance des algorithmes mis en oeuvre vis-à-vis des méthodes existantes, ce cas étant largement étudié dans la littérature, du fait de sa relative simplicité. Celui-ci met également en évidence la robustesse empirique des lois de contrôle obtenues par renforcement, héritée de la logique d'essai-erreur mise en oeuvre lors de l'apprentissage. La montée en Reynolds se fait par le passage à un écoulement bi-dimensionnel décroché autour d'un profil NACA-0012 à un nombre de Reynolds de 1000. Cet écoulement similaire vise à évaluer les capacités de contrôle du DRL concernant un phénomène de décrochage où les instationnarités peuvent être sources de fatigue structurelle pour les véhicules volants. Ces cas permettent aussi la mise en évidence de la capacité exploratoire du DRL comparé aux méthodes existantes. Ces dernières re-

posant majoritairement sur des modélisations linéaires des systèmes à contrôler, proposent des lois de contrôle elles aussi linéaires, alors que l'apprentissage par renforcement s'affranchit de cette hypothèse de linéarité et permet d'explorer un espace de solutions non-linéaires et potentiellement plus pertinentes.

Ces deux derniers cas mettent en évidence des instabilitées globales et prouvent la capacité de l'approche par renforcement à les contrôler. Le cas d'écoulement au-dessus d'une cavité (bidimensionnelle) permet d'étendre ce test à des cas présentant également des instabilités convectives. Ici l'écoulement présente une couche de mélange où se développent des instabilités de Kelvin-Helmholtz. Celle-ci peut-être vue comme un amplificateur de bruit, dont la nature diffère des cas précédemment discutés. Ce cas est incontestablement le plus complexe à contrôler puisque cette dernière caractéristique le rend très sensible au bruit auquel l'algorithme d'apprentissage a recours. Il présente en outre deux mécanismes qui entretiennent l'instabilité, à des fréquences très différentes. Cet écoulement a ainsi permis de mettre en évidence bon nombre des difficultés à surmonter pour le contrôle des écoulements par renforcement. Cette approche a été comparée à des approches génétiques reposant sur des meta-heuristiques calculées sur une population de lois de contrôle plutôt que sur une seule et unique loi. Le potentiel du transfer-learning, comme outil permettant de s'affranchir partiellement des difficultés identifiées, a également été mise en évidence au travers de tentatives préliminaires de transfert entre agents ayant des entrées (signaux de mesure) différentes.

D'une part, ces études ont permis de mettre en évidence des caractéristiques intrinsèques aux écoulements telles que la réceptivité au bruit de forçage ou les délais, dus à la convection, entre une action de forçage et ses conséquences observables. Elles ont aussi rendu possible l'identification de propriétés extrinséques telles que le coût de calcul ou la modélisation des actionneurs, ces dernières étant directement liées à la méthode de modélisation des écoulements. Ces caractéristiques, comparées à celles des environnements traditionnellement utilisés pour qualifier les algorithmes d'apprentissage par renforcement, sont spécifiques à la mécanique des fluides et constituent autant de défis à relever pour transposer avec succès le paradigme de l'apprentissage par renforcement à des applications industrielles ou scientifiques complexes.Cette thèse a permis également de développer une reflexion relative à l'optimisation du nombre et de l'emplacement des entrées (capteurs) et sorties (actionneurs) utilisées pour les lois de contrôle apprises par renforcement. Cette reflexion décrit tout d'abord la nature de la problématique dans le contexte du DRL ainsi que les hypothèses formulées pour rendre la problème tractable. Elle se poursuit par le développement de divers algorithmes en sur-couche du processus d'apprentissage par renforcement visant à réduire le nombre de capteurs et d'actionneurs tout en préservant la performance de contrôle le plus possible. Les résultats des tests des méthodes proposées sur différents cas d'étude sont enfin utilisés pour suggérer des pistes d'amélioration ainsi que pour compléter la compréhension de cette problématique complexe mais néanmoins importante en vue de transposer les méthodes de DRL vers des contextes expérimentaux ou industriels.

DOCTORAL SCHOOL

Titre: Potentiel et défis de l'apprentissage par renforcement pour le contrôle des écoulements

Mots clés: Contrôle des Écoulements, Apprentissage par Renforcement, Mécanique des Fluides Numérique

Résumé: Cette thèse évalue le potentiel de nouvelles méthodes d'apprentissage par renforcement appliquées au contrôle des écoulements. Concernant la mécanique des fluides, les méthodes traditionnelles de contrôle reposent généralement sur des hypothèses fortes de linéarité qui limitent souvent la performance des lois de contrôle obtenues. L'apprentissage par renforcement associé à des méthodes d'apprentissage profond propose de s'affranchir de ces contraintes dans le but d'optimiser des lois de contrôle efficaces, économes en énergie et robustes.
De nombreux défis, spécifiques au contrôle des écoulements, restent à relever pour permettre le développement de telles méthodes dans des contextes expérimentaux et industriels. Contrairement aux environnements sur lesquels les algorithmes d'apprentissage par renforcement sont évalués, la mécanique des fluides met en jeu des systèmes ayant une grande dimensionnalité, un comportement généralement non-linéaire et une observabilité partielle, que ce soit dans un contexte numérique ou expérimental.
Cette étude vise donc à identifier ces problèmes et les conséquences qu'ils entraînent sur les lois de contrôle et leur apprentissage ainsi qu'à proposer de nouveaux algorithmes construits en surcouche de méthodes existantes dans le but de contourner ces obstacles. La plupart de ces contraintes se ramènent à une problématique de coût d'acquisition de la donnée d'entraînement et d'efficacité de l'apprentissage, qui sont des facteurs décisifs quant à la faisabilité et l'efficacité de ces méthodes de contrôle. Les efforts concernant la réduction du nombre de capteurs et d'actionneurs ainsi que l'amélioration de l'efficacité de l'exploration de l'espace d'états conduisent à proposer des modifications des algorithmes d'apprentissage existants ou des méthodes entièrement nouvelles visant à accélérer l'apprentissage.

Title: Potential and challenges of reinforcement learning for flow control

Keywords: Flow Control, Reinforcement Learning, Computational Fluid Dynamics

Abstract: This thesis evaluates the potential of novel reinforcement learning methods applied to flow control. While, for fluid mechanics, state-of-the-art control generally relies on strong linear assumptions that often limit the reach of control laws, reinforcement learning associated with deep learning methods propose to break free from these constraints in order to derive effective, energy efficient and robust control policies.
Still, numerous challenges, coming from the specificity of flow control, are yet to be overcome in order to enable the development of such methods in experimental and industrial contexts. Contrary to the traditional test-bench environments on which state-of-the-art reinforcement learning methods are evaluated, flow control involves a large dimensionality, a generally non-linear behavior and a partial observability, whether it is in a numerical or experimental context.
This study thus aims at identifying these issues and the consequences they yield on training control policies for flow control and to propose novel algorithms built on-top of training methods that help circumvent these problems. Most of these come down to sample cost, i.e. the computational cost of acquiring training data, which is a major decision factor concerning the feasibility and the success of these control methods. Efforts concerning the reduction of both sensor and actuation layouts as well as the improvement of the state exploration efficiency give rise to proposed modifications of existing training algorithms or entirely novel methods aiming at accelerating training.

Institut Polytechnique de Paris
91120 Palaiseau, France