A PH.D THESIS IN COTUTELLE BETWEEN

L'ÉCOLE NATIONALE SUPÉRIEURE
DE TECHNIQUES AVANCÉES BRETAGNE
AND
FLINDERS UNIVERSITY
COLLEGE OF SCIENCE AND ENGINEERING

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Automatique, Productique et Robotique*

By
Thomas CHAFFRE

Reinforcement Learning and Sim-to-Real Transfer for Adaptive Control of AUV

This thesis has been approved by the Dean of Graduate Research on 06 March 2023
Research Unit : Lab-STICC UMR CNRS 6285 - Centre for Maritime Engineering Flinders University

Reviewers before defense :

Tirthankar Bandyopadhyay	Senior Researcher at CSIRO, Australia
David Filliat	Professor at ENSTA Paris and Director of the CIEDS, France

Composition of the jury :

President :	Cédric Buche	Professor at ENIB and IRL CROSSING
Examiners :	Isabelle Fantoni	Research Director at CNRS, member of LS2N, France
	Vincent Creuze	Professor at Université de Montpellier
Thesis co-directors :	Benoît Clement	Professor at ENSTA Bretagne
	Karl Sammut	Professor at FLINDERS University

Guests :

Gilles LE CHENADEC	Associate Professor at ENSTA Bretagne
Paulo SANTOS	Associate Professor at FLINDERS University
Estelle CHAUVEAU	Lead Research Engineer at NAVAL GROUP

Declaration

I certify that this thesis:

1. does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university

2. and the research within will not be submitted for any other future degree or diploma without the permission of Flinders University and or ENSTA Bretagne; and

3. to the best of my knowledge and belief, does not contain any material previously published or written by another person except where due reference is made in the text.

Thomas Chaffre, 06 March 2023.

Remerciements

Je tiens tout d'abord à remercier mon père Julien et ma mère Frédérique qui m'ont énormément soutenu lors de ces trois dernières années, et de façon générale tout au long de ma vie. Ils ont toujours sur m'écouter, me redonner le moral, et me conseiller lors des moments difficiles. Merci d'avoir supporté mes périodes de stress et d'angoisse, de m'aider à prendre du recul et d'avoir directement et indirectement participé à ma thèse. Sans leur soutien mon expérience de thèse aurait été totalement différente. Je tiens également à remercier ma soeur Julie et son mari Dimitri qui m'auront également soutenu durant toutes ces années. Merci pour votre présence, nos échanges et votre joie de vivre qui m'a si souvent permis de relativiser face aux épreuves de la vie. L'affection et les encouragements constants de ma famille m'ont permis de traverser avec confiance cette étape de ma vie.

Je tiens ensuite à remercier mes encadrants de thèse, Benoit Clement, Karl Sammut, Gilles Le Chenadec, Paulo Santos et Estelle Chauveau, sans qui cette thèse n'aura pas eu lieu. Durant ces trois dernières années j'ai pu bénéficier d'un encadrement et d'une disponibilité sans failles de leur part. Je vous remercie d'avoir cru en moi et de m'avoir donné cette opportunité de m'exprimer et de me développer intellectuellement. J'ai pu grâce à vous apprendre tout ce qu'implique le travail de recherche et pour cela je vous en suis extrêmement reconnaissant. Travailler avec vous durant ces années fut un réel plaisir et je garderai en mémoire nos longues discussions. Je vous suis reconnaissant pour votre bienveillance, votre soutient appuyé lorsque j'en avais besoin et pour m'avoir accordé une pleine liberté dans mes recherches.

Je tiens tout particulièrement à remercier Tony Kyriacou qui aura tout fait pour que mon séjour Australien à Flinders University se passe le mieux possible. Tony est une personne formidable qui m'a permis de me sentir comme à la maison et ce dès mon premier jour au campus. Tony m'a également beaucoup aidé pour mes démarches administratives et pour toutes ces raisons je tenais encore à le remercier.

Je tiens à remercier Jonathan Wheare et Andrew Lammas, ingénieurs de recherche à Flinders University, qui m'ont aidé à mettre en place nos validations expérimentales. Je les remercie pour leur temps et leur patience au fil de ces mois d'expérimentations pendant lesquels ils ont dû s'adapter à de nombreux imprévus. Sans eux, nous n'aurions pas pu obtenir ces résultats. J'ai pu à vos côtés me rendre compte à quel point la robotique mobile est une tâche compliquée et qu'il faut s'attendre à tous les scénarios avec ce type de véhicule.

Merci à l'équipe de l'IRL CROSSING d'Adelaide avec qui j'ai pu travailler et où j'ai pu me ressourcer lorsque le mal du pays se faisait sentir. Merci Jean-Philippe et Cédric ainsi que toutes les personnes de votre équipe pour ces moments.

Et pour finir, la vie se continuant en dehors du laboratoire, je tiens à remercier mes amis, merci Gabriel, Armagan, Ludovic, Maëlic, Thomas, Yoann, Quentin, Joris, Julien, Carlos et Alam.

Summary

Autopilots for unmanned systems are usually designed based on the feedback provided by velocity and orientation sensors. In the case of autopilot systems for autonomous underwater vehicles (AUVs), the main objective in the design is to compensate for waves and current-induced disturbing forces acting on their body. Existing AUV autopilots are however only able to compensate for low-frequency components of sea-induced disturbances. It seems natural to assume that the AUV performance could be improved by taking the nature of the disturbances into account in the design of the autopilot.

Adaptive control provides what seems to be an ideal framework for this end. The objective of this technique is to adjust automatically the control parameters when facing unknown or time-varying processes such that the desired performance threshold is met. Developed in the late 1950s, adaptive control frameworks have been considerably expanded and used in various fields, their application has been facilitated by the rapid progress in microelectronics and the increasing interaction between laboratories and companies, from aerospace to maritime industries. As a result, adaptive controllers started to be widely adopted in the industry in the early 1980s. It was established at that time that *robust designs* with fixed parameters are too limited to handle complex regimes. The study of adaptive controllers for AUV maneuvering is associated with various challenges, and the focus of this thesis was the external disturbances including:

Unknown dynamics: the uncertainty associated with describing precisely the states of waves or currents is high. This, together with its dynamic nature, prevents linear feedback control methods from achieving optimal performance of the plant. This becomes more critical in the presence of changes in weather conditions that impose a multiplicative factor in the component of the induced forces. The disturbance period will also vary with the speed of the vehicle and its orientation relative to the waves.

Nonlinearity: the controller response at some operating points must be overly conservative to satisfy the specification at other operating points. This is difficult to achieve for fixed parameters obtained through local linearization, that do not encompass the entire regime envelope.

In this thesis, we considered the case where the AUVs have limited observability of the process and therefore the aforementioned uncertainties are not measured by the system. A class of adaptive control methods, known as learning-based adaptive controllers, have been developed to tackle some of these limitations. This family of solutions uses model-free optimization methods capable of compensating for the unknown part of a process while also maintaining optimal control of its known part using traditional model-based control structures. Among the various model-free methods, deep reinforcement learning is currently leading the field. They exploit strong statistical tools that provide control systems the ability to automatically learn and improve from experience without being explicitly told how to.

The objective of this thesis was to formalize a novel learning-based adaptive control using deep reinforcement learning and adaptive pole-placement control. In addition, we proposed a novel experience replay mechanism that takes into account the characteristic of the biological replay mechanism. The methods were validated in simulation and in real life, demonstrating the benefits of combining both theories against using them separately.

Contents

1 Introduction

1.1 Context and motivations

Autonomous vehicles are becoming increasingly prevalent in our day-to-day activities. From cars, trains, warehouse robots to medical delivery quadcopters, the field of autonomous vehicles is blooming. This development was driven by an urge to strengthen productivity, accuracy, operational efficiency but also to improve human operators' and users' safety. While this tendency is seen globally, there is also evidence of an unequal development in underwater applications. Despite the comparable needs in applications, including offshore platforms inspection, marine geoscience, harbor and coastal surveillance and underwater mine countermeasures, the majority of uncrewed underwater vehicles remain remotely operated or with limited autonomy abilities..

The problem of autonomy is even more pronounced in the context of small-size underwater vehicles. The latter are paradoxically denoted as AUVs for Autonomous Underwater Vehicles, not on the basis of improved autonomy, but rather by dint of the historical need for physical connection to a surface vessel that has been removed. AUVs are required to operate over large regions (from deep oceans to coastal and riverine regions), lengthy periods of time (extending from several hours to days before the possibility for human intervention) and to perform complex tasks such as search and rescue [AC05], underwater manipulation [MCY09], pipeline and facility inspection operations [GNO12], under-ice exploration [Bar+20], target following [Sun+15], etc. Although there has been rapid development of aerial and terrestrial autonomous vehicles, there has been relatively slow development of AUVs, leading to interrogations as to what could be the cause of this contrast.

Until the late 1940s, it was widely believed that fixed control design was sufficient for processes of limited uncertainties. It was only with the development of autopilots by NASA for their high-performance aircraft, such as the X-15 in the early 1950s, that it was recognized that constant-gain linear feedback control could not provide satisfactory performance over the entire flight regime. Such aircraft operate over a wide range of altitudes and speeds with their internal dynamics changing as fuel is consumed. A more sophisticated control system able to maintain its performance over multiple operating conditions was therefore needed. Great efforts were therefore made into the development of autopilots for the X-15, which led to the emergence of what is known today as Adaptive Control. The objective of adaptive control is to grant the control law some flexibility to adjust its response based on process variation. The Gain Scheduling technique [DAL10] was then proposed as a solution for the control of hypersonic aircraft, becoming shortly after a standard tool in the field. Following this successful application, adaptive control has since attracted significant attention from both academic and industry communities. Nevertheless, the deployment of autonomous vehicles in the maritime domain was still limited. Adaptive control methods require some a priori knowledge of the processes and the system to be controlled and often require expensive simulation efforts. This first generation of methods is denoted as model-based [Kál58] [Bel15][MBT59] accordingly due to the need for a model of the process dynamics to be designed. The difference observed in maritime vehicles can be explained in part due to the limited measurement abilities on board the vehicles and our little to no understanding of the natural phenomena taking place underwater and disturbing the vehicles. This lack of prior knowledge is even more problematic in the case of AUVs, which in addition face several challenges [Has+16], including:

1. *Unknown dynamics*: the uncertainty associated with describing precisely the states of waves or currents is high. This, together with their dynamic nature, prevents linear feedback control methods from achieving optimal performance of the plant. This becomes more critical in the presence of changes in weather conditions that impose a multiplicative factor in the component of the induced forces. The disturbance period will also vary with the speed of the vehicle and its orientation relative to the waves.
2. *Nonlinearity*: the controller response at some operating points must be overly conservative in order to satisfy the specification at other operating points. This is hardly possible for fixed parameters obtained through local linearization, which does not encompass the entire regime envelope.
3. *Thruster efficiency*: a fully-actuated vehicle can often become underactuated when its speed varies. This is especially true for hovering-type AUVs which use thrusters in place of steering fins to achieve maneuverability at low speeds. As the forward speed increases, the effectiveness and efficiency of lateral thruster-induced movements are drastically reduced, making it impossible for the vehicle to account for pure lateral motions.
4. *System reliability*: if the performances of one or more thrusters become increasingly less effective, the control system should be able to detect this and engage a new control algorithm specially designed to accommodate the failures and, if possible, to complete the mission.

The design of autopilots under these circumstances seems laborious. As a result, the vast majority (95%) of AUV control systems are based on fixed PID controllers, despite their poor performance against process variation [Yuh00]. A new class of adaptive controllers is desired to reduce or remove this need for prior knowledge. When no information on the process is available, model-free adaptive control can be used. These methods aim at describing complex processes from sensory data only, rather than from first principle laws. The first such theories appeared in French literature in the early 1920's [Leb22]. The concept was then formalized under what is currently known as *Extremum Seeking Control* (ESC) [Sut19]. The goal of ESC is to maximize a control objective function without any knowledge of its extremum. Since its initial development, research in model-free adaptive control theory has gained pace, exhibiting a reliable level of maturity with good analysis and understanding of its main properties. One of the main applications of ESC has been in reaction processes, including fuel flow control, combustion process control, and wind or solar energy conversion.

However, the lack of guaranteed transient performance remains a significant drawback in the application of model-free methods in AUV control. This is due to the safety of the platforms which is most important given the cost of deploying such vehicles. This led to the development of an entirely new theory of adaptive control, denoted as learning-based. It is based on the idea that by combining both model-based and model-free theories, one designer can compensate for both the known and unknown parts of the process model. The concept behind this theory is to use a model-based control structure to maintain optimal control of the known part of a process, and to compensate for the unknown part by learning to adapt to process variations in a model-free fashion. Following the recent breakthrough in data-driven control [BK19] [Ben18] and the successful development of modern machine learning techniques [SB18] with improvements in computing power (e.g. faster GPUs, enhanced Parallel computing, etc), we have seen a resurgence of interest in the field of learning-based adaptive control.

In particular, such methods based on deep reinforcement learning (DRL) have shown promising achievements. This is possible, especially thanks to the use of deep neural networks to extract physical insights through sensory data (empowered by progress in data collection with high-fidelity simulations, faster computers, etc). However, as stated in [Sün+18], the extension of DRL techniques for robotic tasks raises serious questions. Compared to other applications, robots have to interact with a dynamic environment where relevant information about the system is not always accessible or tractable over time.

In their up-to-date thorough investigations [Dul+20], G. Dulac-Arnold *et al.* presented real-world reinforcement learning challenges that are still not resolved, these include *Satisfying Environmental Constraints, High-Dimensional Continuous State and Action Spaces or Multi-Objective Reward Functions*. In addition, the notions of stability (in terms of Lyapunov stability) in the DRL framework are more deeply investigated [GF15] but the lack of formalism is still often highlighted by the control community as a significant concern. The purpose of this thesis is to explore to what extent model-free control techniques combined with the classical theory of dynamical systems can be, at least, part of the solution to these challenges. This is the objective of the learning-based control area [Ben14] which aims to combine the advantages of the aforementioned paradigms (model-based and model-free) into one hybrid control scheme.

An adaptive controller, being by nature nonlinear, is more complicated and thus more computationally expensive than a fixed-gain controller. Therefore, before choosing to use an adaptive controller it is important to study whether or not the problem could be solved by a constant-gain controller. One way in deciding if an adaptive controller is justified is outlined in Figure 1.1 depending on the characteristic of the disturbances. Following this procedure, we discuss next why adaptive control is mandatory in our use case of AUV control.

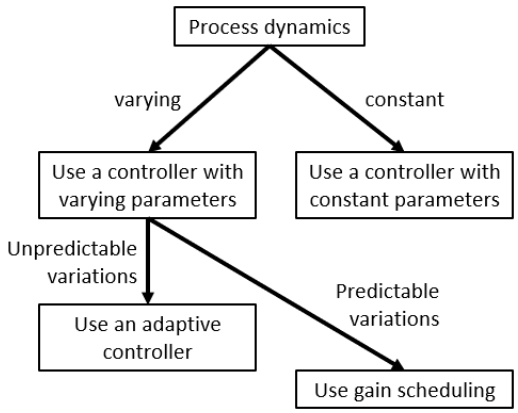


Figure 1.1: *Procedure to decide what type of controller to use.*

Open vs closed-loop control

Although the act of control seems active, there are many examples of control in everyday life that is not active but rather passive. An example is the hull of boats that passively causes the water around the body to behave in a favorable way to reduce drag. If we can reach the desired behavior with passive control, then we just have to design beforehand the vehicle's body properly and there will not be energy expenditure. Passive control, however, is typically not enough and often some form of active control is required. Active control essentially means that we are injecting energy into the system to actively manipulate its behavior. There exist many solution methods for active control. The most common form of active control is called open-loop where, as illustrated in Figure 1.2, we reverse design the controlled system and invert the dynamics in order to determine exactly what is the optimal control input u to get the desired output y. The principal downside of open-loop control is that we are always putting energy into the system and the moment we stop, the system becomes unsafe. What we can do instead is to take sensor measurements of what the system is actually doing, then somehow build a controller, and feed that back into the input signal so that can manipulate the system. This allows much more subtle control of the system with very low energy input. The basic idea of closed-loop feedback control is that by measuring the system output, we can often do much better than just feeding in pre-planned control input, and this is the entire subject of this thesis.

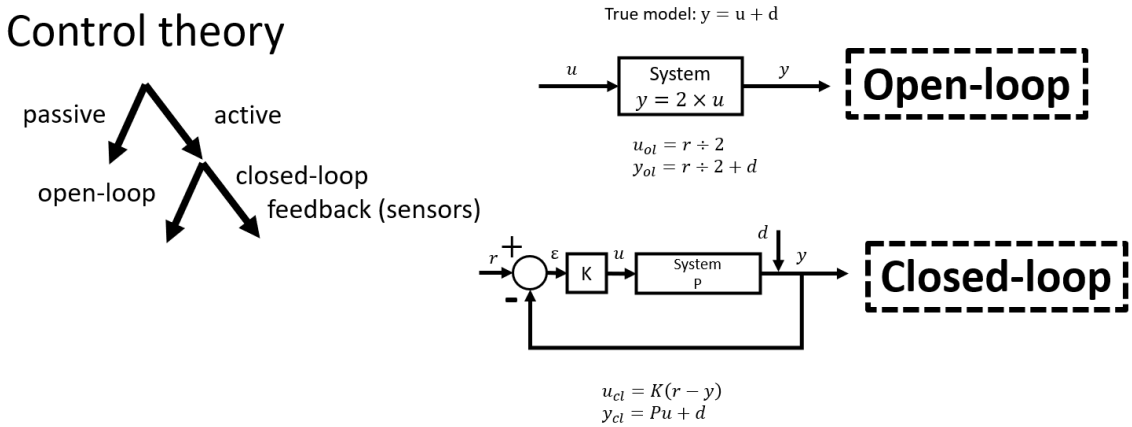


Figure 1.2: *Control and dynamics are two sides of the same coin that are present in every aspect of our life. Control can be either passive or active, and allows us to ensure that our system of interest behaves as desired. The most common form of active control is known as closed-loop feedback control where the system output is measured and used to compute the next control input. As illustrated here, the closed-loop feedback controller is much more efficient at handling uncertainties and disturbances compared to its open-loop counterpart.*

In order to show how we can reduce the effect of disturbances, let's now illustrate the benefits of feedback with a simple example. Consider an AUV facing a current disturbance in its environment and we want to control its speed. Consider now that we are given a nominal model for the system that is a very crude model of the AUV such as:

$$y = 2u. \tag{1.1}$$

We do not give a unit to this value (1.1) but let's just say we measured, in a test tank without disturbance, that by increasing the input to the thrusters by one unit, we obtain an increment of two units of velocity. Let's see now how open-loop control would solve here a tracking problem where we want the AUV to maintain a reference velocity value r.

If we want the system output y to be equal to r, with open-loop control u_{ol} we will have:

$$u_{ol} = \frac{r}{2}. \tag{1.2}$$

This (1.2) is a very bad controller for many reasons, namely uncertainties in the model and disturbances. In fact, let's imagine that the model would actually be $y = u$ (possibly because the vehicle's thrusters are old, the sensors are slightly out of calibration, and maybe the drag has changed, etc, since the vehicle was originally modeled). Most likely, the vehicle is no longer fully efficient and y is actually just equal to u, so it is only half as responsive as our nominal model (1.1). In this case, the open-loop controller (1.2) may not achieve the desired velocity r, but instead, only achieve $\frac{r}{2}$. If the actual system differs from the nominal model, the open-loop controller has no way of correcting that. Moreover, if we add the disturbance, the true model y_T of the AUV becomes:

$$y_T = u + d, \tag{1.3}$$

with d a scalar representing the disturbance. In this case, the open-loop controller (1.2) is only tracking 50% of the reference velocity and all of the disturbance passes right through to the actual velocity y. Open-loop

control does not take into account the uncertainties and disturbances at all and therefore fails at controlling the AUV in this context.

On the other hand, as illustrated in Figure 1.2, we can close the loop with some controller K. The idea is to measure the actual velocity of the AUV (so we can tell if it is speeding up or down), feed that back, and subtract it from the reference velocity (the difference is an error signal ε). Feedback control then consists in choosing K to make ε small. This is denoted as proportional feedback control because K is just a number. The closed-loop feedback controller is thus defined as:

$$u_{cl} = K\varepsilon = K(r - y), \tag{1.4}$$

and the resulting model of the closed-loop feedback control is:

$$y_{cl} = Pu + d. \tag{1.5}$$

We can solve y_{cl} as a function of r and d as:

$$\begin{aligned} y_{cl} &= Pu + d \\ &= PKr - Pky_{cl} + d \\ (1 + PK)y_{cl} &= PKr + d \\ y_{cl} &= \underbrace{\frac{PK}{1 + PK}}_{tracking} r + \underbrace{\frac{1}{1 + PK}}_{disturbance} d, \end{aligned} \tag{1.6}$$

where the left term tells us how well the output velocity y_{cl} matches the reference velocity, and the right term tells us how much disturbance gets reduced by the control. For recall, the nominal model is $P = 2$ (1.1), but the true AUV system has $P = 1$. Notice here (1.6), we want to have the left term equal to 1 which means that we have a really good reference tracking. We want the right term to be equal to 0 because if we have disturbance, we want whatever feedback multiplying it to be as small as possible. We can directly see that the best way to achieve that is to consider a very big value of K. For instance, with $K = 100$, the left term in (1.6) is very close to 1, the actual velocity is only off from the reference velocity by about 1%, and we have reduced disturbance by about a factor of a 101. In contrast, the open-loop controller was off by 50% because the model was bad, and the disturbance was not reduced at all. This is an example where we can handle model uncertainties and disturbances by taking measurements and feeding them back in proportional control. There exist various techniques to choose the value of K and optimal control theory [KS72][Ber95][Doy96] is the most common method traditionally used for this purpose.

Nevertheless, when facing process variation, the resulting fixed value of K is not enough to satisfy control performance over a wide spectrum of operating conditions. When the physics of the process is reasonably well known, it is possible to determine suitable values of K for different operating conditions by linearizing the models. System identification is an alternative to physical modeling, however, both approaches do require a significant engineering effort in addition to knowing the a priori process model.

In this thesis, we consider the case where these approaches can not be used because we do not have access to a model process and the disturbances are not measured. In particular, we are focusing solely on external disturbances that are sea currents. In the following, we illustrate this limit and therefore the need for adaptation.

Limits of fixed optimal control

Effect of process variation: The standard approach to control system design has been to develop a linear model for some operating conditions and to design a closed-loop feedback controller having constant parameters. This approach has been remarkably successful as we have been able to see that such controllers are intrinsically insensitive to modeling errors and disturbances. However, in practice, there are many different sources of process variations, and the underlying reasons for those variations are not fully understood. We will consider here two simple examples to illustrate the difficulties rising from process variations. Consider a closed-loop feedback controller with a Proportional and Integral term, a nonlinear AUV, and the third order process $G(s)$ illustrated in Figure 1.3. Let's consider the characteristic of the AUV as follows:

$$v = f(u) = u^4,\ u \geq 0. \tag{1.7}$$

When linearizing the system around a steady-state operating point, we can see that the incremental gain of the AUV is $f'(u)$ (i.e. the loop gain is proportional to $f'(u)$). This means that the system can perform well at one operating point and poorly at another. This is illustrated in Figure 1.4 where we tuned the controller parameters using optimal control theory to have limited oscillation at the operating condition $U_c = 0.01$. For higher values, the controller becomes unstable as shown in Figure 1.4d.

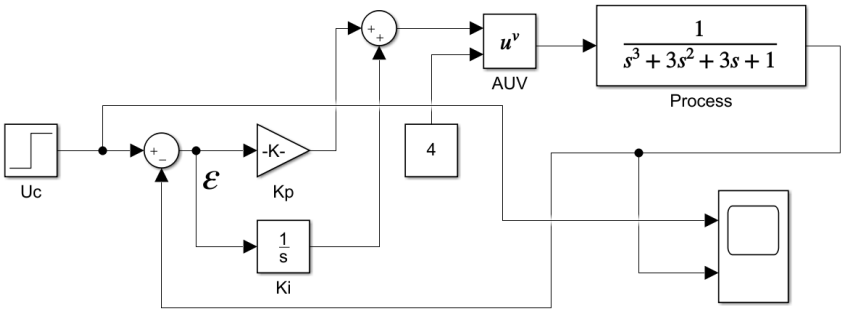
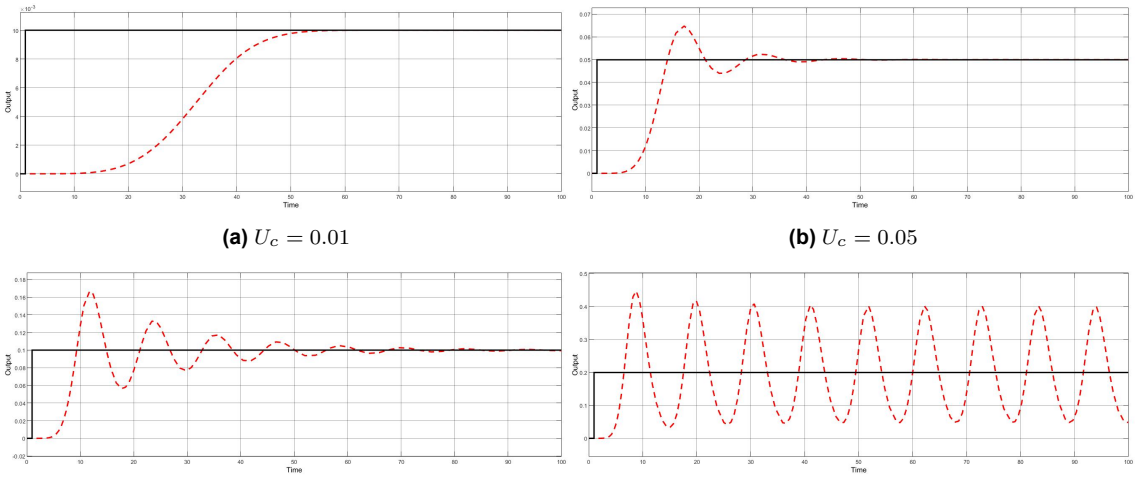


Figure 1.3: *Block diagram of an AUV control loop with a PI controller.*



(a) $U_c = 0.01$ **(b)** $U_c = 0.05$

(c) $U_c = 0.1$ **(d)** $U_c = 0.2$

Figure 1.4: *Step responses for PI controller at different operating conditions. The controller parameters are* $K = 0.15$, $T_i = 1$, *the AUV characteristic is* $f(u) = u^4$ *and* $G_0(s) = 1/(s+1)^3$.

Effect of disturbance variation: The second issue to be explored in this thesis is the effect of variations in current disturbance characteristics. We propose to model this disturbance as an additional disturbing force as illustrated in the block diagram displayed in Figure 1.5. Again, we tuned the controller parameters using optimal control theory for the same operating condition $U_c = 0.01$ and we plot the controller response against the different values of sea current denoted by d.

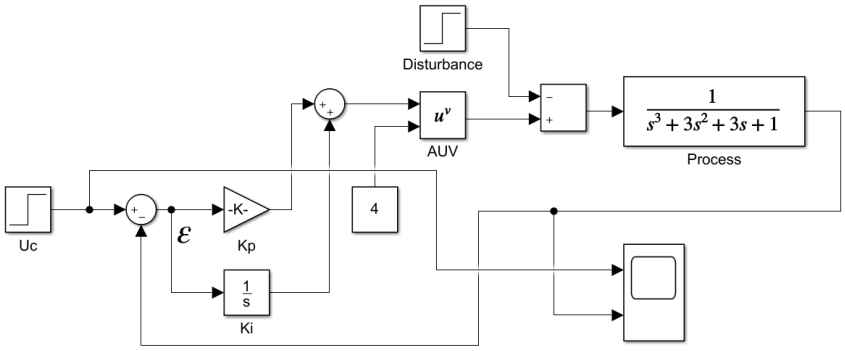
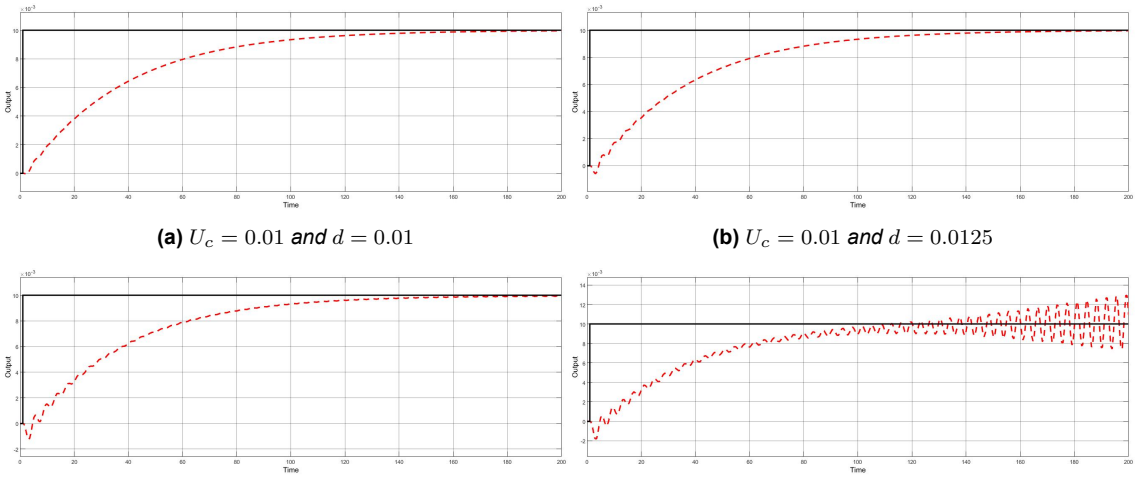


Figure 1.5: *Block diagram of an AUV control loop with a PI controller.*



(a) $U_c = 0.01$ *and* $d = 0.01$

(b) $U_c = 0.01$ *and* $d = 0.0125$

(c) $U_c = 0.01$ *and* $d = 0.0150$

(d) $U_c = 0.01$ *and* $d = 0.0175$

Figure 1.6: *Step responses for PI controller at different operating conditions. The controller parameters are* $K = 31$, $T_i = 1$, *the AUV characteristic is* $f(u) = u^4$ *and* $G_0(s) = 1/(s+1)^3$.

As illustrated in Figure 1.6, the optimal values of K and T_i that were obtained for the operating condition $U_c = 0.01$ and $d = 0.01$ are no longer satisfactory when the value of the disturbance change. Over a certain value of d, the controller is even diverging from the steady state as shown in Figure 1.6d. It seems intuitive that we can recover from these process variations by adjusting the controller parameters according to the operating conditions with adaptive control. However, in the AUV case, we do not have direct access to disturbance measurements and we do not have access to a model of its effect on the AUV. Thus we can not perform this tuning procedure beforehand to find optimal values of the control parameters based on the process variations.

Now that the challenge of AUVs control in our context is demonstrated, we present next the problem and the research objectives of this thesis.

1.2 Problematic and research objectives

As previously discussed, the design of an adaptive controller for AUVs in our context is associated with a number of challenges. The principal problem is that the AUV does not have a sensor to directly measure the current and wave disturbances. While a downward-looking doppler velocity logger (DVL) can be used to measure a current below the vehicle, this only works when the DVL has a bottom lock, i.e, the seabed is within acoustic range of the DVL. As a consequence, we have limited sensorial ability (we only have access to the IMU feedback), and therefore the disturbance is completely ignored in the process modeling, preventing us from using standard model-based adaptive control methods. The overall objective of this thesis is to propose a learning-based adaptive control system able to compensate for the current disturbance despite not measuring it directly. This design is associated with several research questions that we aim to study in this thesis, including:

- What is the **impact of process observability** on the design of **model-based** controller for AUVs?
- What are the **most effective strategies** to use the deep reinforcement learning **as a model-free** tuning method in adaptive control systems?
- How do the **combination** of model-based and model-free theories compare in regularization performance and robustness to disturbance as opposed to when **exploited separately**?
- What effect does the **experience replay** mechanism have on the **learning dynamics** and the **generalization ability** of the resulting policy?
- What are the **similarities and differences** in the sim-to-real transfer of **control parameters and control inputs?**

The number of AUV applications is increasing fast and control designers are obliged to address more and more complex tasks while being limited in the complexity of the control laws they can use due to restricted power-conservative onboard computational resources coupled with the issue of insufficient knowledge of underwater dynamics. To cope with this situation, the design and development of a learning-based adaptive control system conceived on the basis of a model-based control structure adjusted by a model-free optimization algorithm would allow (1) to compensate for the known part of the process with strong stability components, and (2) to effectively compensate for unobservable external disturbance. The principal objective of this thesis is to design a learning-based adaptive control system that could adapt to changes in process dynamics and current disturbance characteristics. The research objective is presented, and we discuss next the proposed structure of the thesis which aims to answer the aforementioned questions.

1.3 Structure of the thesis

By way of introduction, the first section presents the context of the thesis with the particularities of AUV applications. Then, the general problem of AUV control is introduced with the objectives of the thesis. Then, the research questions are defined and finally, the research methodology is described.

Section 2 presents the technical background regarding the building elements of AUV adaptive control. The first part is dedicated to the technical elements of adaptive control. We propose a classification of solutions methods based on the dependence on the process model resulting in three classes of adaptive control methods. A succinct description of each of these methods is provided with a greater emphasis given to learning-based methods. This part ends with a discussion of the limitations of these solutions to our problem of AUV control. In the second part, we present the background elements of reinforcement learning theory. Classification of solutions methods is provided in terms of the nature of the action sampling methods. Then a complete description of the maximum entropy deep policy gradient method is provided, which is used as a central building block of the proposed control system. This part ends with an analysis of the limitations of deep reinforcement learning (DRL). Finally, this second section ends with a description of the simulation tool used to train the aforementioned algorithms.

In Section 3, the research steps that lead to the proposed control system design are presented. It starts with the first part comprising a literature review of related works in learning-based adaptive control of AUVs. Then, in the second part, we present the preliminary studies that were motivated by this literature review which include: the study of a model-free and a learning-based adaptive control and stability analysis. Finally, the section ends with the proposed learning-based adaptive control system.

In section 4, the proposed control system is evaluated and validated under simulation. It is applied to different AUV applications and improved designs are provided throughout the section. It ends with some additional findings and insights that will lead to the design of the sim-to-real transfer methodology.

Section 5 is devoted to the experimental validation of the proposed method. It starts with a description of the experiment protocol and settings. Then, the design of the control system is presented along with an adjusted domain randomization procedure. The section ends with an analysis of the experimental results.

In the General Conclusions and Perspectives section, we come back to the contributions of this thesis to the field of AUV adaptive control. We will finally present different perspectives on improvement.

1.4 List of contributions

Publications

Thomas Chaffre, Jonathan Wheare, Andrew Lammas, Paulo Santos, Gilles Le Chenadec, Karl Sammut, Estelle Chauveau, and Benoit Clement (2022). Sim-to-real transfer of adaptive control parameters for improved robustness to sea current variations, under review.

Thomas Chaffre, Paulo E. Santos, Gilles Le Chenadec, Estelle Chauveau, Karl Sammut, and Benoit Clement (2022). Learning Stochastic Adaptive Control using a Bio-Inspired Experience Replay Experience Replay. In TechRxiv.

Hector Kohler, Benoit Clement, Thomas Chaffre, and Gilles Le Chenadec (2022). PID Tuning using Cross-Entropy DeepLearning: a Lyapunov Stability Analysis. In *Proceedings of the 14th IFAC CAMS*.

Thomas Chaffre, Julien Moras, Adrien Chan-Hon-Tong, Julien Marzat, Karl Sammut, Gilles Le Chenadec, and Benoit Clement (2022). Learning-based vs Model-free Adaptive Control of a MAV under Wind Gust. In *Informatics in Control, Automation and Robotics pp 362–385, LNEE, SPRINGER*.

Thomas Chaffre, Gilles Le Chenadec, Karl Sammut, Estelle Chauveau, and Benoit Clement (2021). Direct Adaptive Pole-Placement Controller using Deep Reinforcement Learning: Application to AUV Control. In *Proceedings of the 13th IFAC Conference on Control Applications in Marine Systems, Robotics and Vehicles (CAMS)*.

Thomas Chaffre, Julien Moras, Adrien Chan Hon Tong, and Julien Marzat (2020). Sim-to-Real Transfer with Incremental Environment Complexity for Reinforcement Learning of Depth-Based Robot Navigation., in *Proceedings of the 16th ICINCO*.

Sola Yoann, Chaffre Thomas, Le Chenadec Gilles, Sammut Karl, and Clement Benoit (2020). Evaluation of a Deep-Reinforcement-Learning-based Controller for the Control of an Autonomous Underwater Vehicle. In *Proceedings of Global Oceans 2020*.

Seminar presentation

Thomas Chaffre. Deep reinforcement learning and transfer of adaptive control parameters for improved robustness to current disturbance. At Centre de Recherche en Automatique de Nancy (CRAN) (2022).

Thomas Chaffre. Learning-based adaptive control of AUVs. At Sirehna at Technocampus Ocean (2022).

Thomas Chaffre, Paulo E. Santos, Gilles Le Chenadec, Estelle Chauveau, Karl Sammut, and Benoit Clement. Learning Stochastic Adaptive Control using a Bio-Inspired Experience Replay Experience Replay. Poster presentation during the visit of Antoine Petit, President and CEO of CNRS, at IRL CROSSING (2022).

Thomas Chaffre. Learning-based adaptive control of AUVs. At IRL CROSSING (2021).

Software

Thomas Chaffre, Hector Kohler. *An environment for learning-based adaptive control of AUVs under Gazebo and ROS*. Github repository used from Section 3 to Section 5 (not open source as IP is shared with ENSTA Bretagne, FLINDERS University and Naval Group). The package includes the implementation of several deep reinforcement learning algorithms and a framework to use them with Gazebo and ROS.

2 Background

This section presents the technical elements necessary to derive the method proposed in this thesis and to support our choice of approaches. First, we explain the need for adaptation when facing process variation and how to design such a control system using adaptive control theory. A classification of solution methods is provided based on the dependence on the process model resulting in three classes of methods: model-based, model-free, and learning-based. In the latter, there exist many machine learning techniques to use as a model-free optimization procedure, and this thesis focuses on the use of deep reinforcement learning. Therefore, we here present the theoretical elements of reinforcement learning and deep policy gradient methods, in particular. To conclude, because deep reinforcement learning methods need to interact greatly with the environment, we finally present the different simulation tools we used to train our algorithms.

2.1 Adaptive control

2.1.1 The adaptive control problem

Interest in industrial adaptive control systems started in the 1950s with the emergence of flight controllers [DAL10]. It was clear at that time that in order to adapt to process variation, one has to exploit any knowledge available to adapt the control system response to changes in operating conditions. To illustrate the adaptive control problem, consider the dynamics of a system described by the nonlinear differential equations of state:

$$\begin{aligned} \dot{x} &= f(t, x, u, p),\ t \in \mathbb{R}^+, \\ y &= h(t, x, u), \end{aligned} \tag{2.1}$$

where $x \in \mathbb{R}^{n_x}$ is the state vector; $u \in \mathbb{R}^{n_u}$ is the vector of control inputs; $p \in \mathcal{P} \subset \mathbb{R}^{n_p}$ is a vector of unknown parameters that is an element of an *a priori* known set $\mathcal{P}$; $y \in \mathbb{R}^{n_y}$ is the vector of system outputs; f and h are smooth functions. The set of outputs and control inputs available at time t is:

$$\Gamma_t = \{y(t), \dots, y(0), u(t), \dots, u(t)\},\ t \in \mathbb{R}^+,\ \Gamma_0 = 0. \tag{2.2}$$

The control performance index historically takes the following form:

$$J = \frac{1}{t} \int_0^t e^2(\tau) d\tau,\ e(t) = w(t) - y(t), \tag{2.3}$$

where $w(t)$ is the desired setpoint. The process is controlled by a controller with adjustable parameters. The general adaptive control problem consists in finding the control policy $u(t) = u_t(\Gamma_t) \in \bar{\Omega}_t$ that minimizes the performance index (2.3) for the system described by (2.1) with $\bar{\Omega}_t$ representing the domain in the space $\mathbb{R}^{n_u}$ where the entire admissible control values are defined. The fundamental hypothesis considered in adaptive control theory [PB61] states that there is a design procedure that makes it possible to determine a controller which satisfies some design criteria if the process and environment are known:

Hypothesis: *For any possible values of a plant model's parameters, there is a controller with a fixed structure and complexity such that the specified performances can be achieved with appropriate values of the controller parameters.*

Accordingly, the task of adaptation is to search for appropriate values of the controller parameters. The adaptive control problem can be summarized as the design of a method capable of adjusting the controller parameters when the characteristics of the process and environment are unknown or changing. This adjustment can be done in two fashions: by *direct* adaptive control, in which the controller parameters are determined without any estimation of the aforementioned uncertainties; by *indirect* adaptive control where the process model, and potentially the disturbance characteristics, are first determined. This emphasizes that *a priori* knowledge about the process and system is required in the adaptive control theory in order to specify achievable performances, to determine the structure and the complexity of the controller as well as the choice of a proper adjustment method. Traditionally, adaptive control methods are classified based on the nature of the process model (i.e. linear vs nonlinear, discrete vs continuous,…). Conversely, we decided to classify the methods based on their dependence on the process model, which fits better the main purpose of this work. Therefore, three classes of adaptive control methods can be identified: model-based [Ste80; DAL10; Kre73; Lan84; Par81; Käl+79], model-free [Leb22; Rot00; KW00; Nes09; Biz20; Sol+20a] and learning-based [WH09; Spo+01; LVV12; Wan+06; KRP06; BA13] approaches. We will present each of these solution methods in the following.

2.1.2 Classification of solution methods

Model-based adaptive control

When the physics of the system is known, it is possible to determine a suitable adaptive controller based on a mathematical modelization of its dynamics. For such systems, one can make an approximation near equilibrium points and then derive a simple differential model using Newton's second law. Adaptive controllers designed entirely in accordance with this kind of representation are denoted as model-based.

One of the first and most straightforward model-based adaptive methods proposed is called *Gain Scheduling*. It consists in finding suitable scheduling variables that effectively characterize the process (on the basis of knowledge about the physical system) and tuning the controller parameters directly based on these variables. This is possible due to the fact that, in many situations, measurable variables can be found that correlate well with changes in the process dynamics. Originally used to adapt changes in the process gains only (hence the name), gain scheduling was first developed to design autopilots for high-performance aircraft [Ste80; DAL10]. As can be seen in Figure 2.1, this method is composed of a linear controller whose parameters are changed as a function of operating conditions in a "preprogrammed" way. After the scheduling variables are determined, the controller parameters are computed at a number of operating conditions based on the system's performance. Gain Scheduling can be seen as a nonlinear mapping from process parameters to controller parameters [Kre73] that is usually framed as a function or a table lookup. Application of Gain Scheduling to maneuvering industrial ships can be found in [VKM96; WD05].

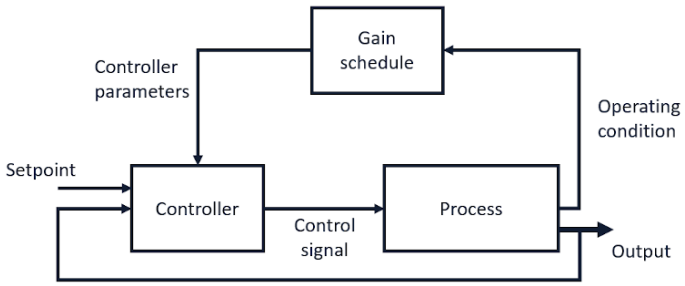


Figure 2.1: *Block diagram of a Gain Scheduling control system. It can be framed as a feedback control system in which the feedback gains are tuned on the basis of the operating conditions.*

The major drawback of Gain Scheduling is that there is no feedback from the closed-loop system to compensate for an incorrect schedule. Its design can be highly time-consuming since the controller parameters must be determined for many operating conditions, and the associated performance needs to be verified by extensive simulations (that are not as straightforward to conduct in the underwater domain as they are for aerial simulations). In addition, in the context of AUVs, we are limited in the number of sensors available, and most of the time, we can not measure most of the scheduling variables of interest. For this reason, Gain Scheduling can not be considered for the adaptive control of AUVs.

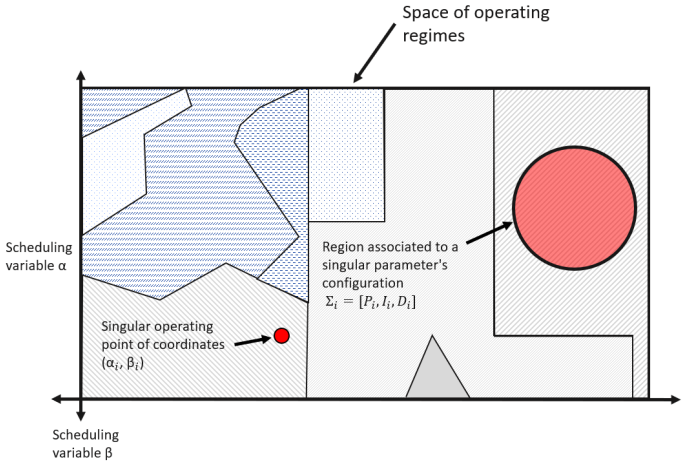


Figure 2.2: *Illustration of the resulting table lookup obtained when using 2 scheduling variables. The space of operating regimes (either discrete or continuous) is divided into regions associated with a particular value of the controller parameters. Depending on the value of the scheduling variables α and β, we switch from one configuration to another. Particular attention must be given to the intersection of operating conditions where nonlinearities exist.*

For problems where the performance specification is formulated in terms of a reference model, Gain Scheduling might not be applicable because of the increasing number of possible scheduling variables and the nonlinearities taking place at the intersection of regimes. The Model-Reference Adaptive System (MRAS) has been developed to tackle this type of problem. It uses a desired reference model to tune the controller parameters by comparing its output y_m to the actual output of the plant y in such a way as to minimize the output error and to maintain the feedback signals bounded. The principal challenge with MRAS is to determine the adjustment rule in order to obtain a stable system that brings the error to zero. The adjustment mechanism presented in the original formulation of MRAS is called the MIT rule $\frac{\mathrm{d}\theta}{\mathrm{d}t} = -\gamma \times \frac{\partial e}{\partial \theta}$, [OWK61], where $e = y - y_m$ is the model error and θ is a control parameter. The quantity $\frac{\partial e}{\partial \theta}$ is known as the sensitivity derivative of the error with respect to θ and γ is the adaptation rate. This adjustment can be regarded as a gradient scheme to minimize the squared error. Various adjustment mechanisms have been proposed for MRAS [Lan84; Par81] and its stability analysis has been thoroughly conducted using Lyapunov theory [Mor79; NV80; GM87]. The MRAS framework has been demonstrated to be suitable for the control of various types of maritime vehicles, from surface tankers [Käl+79] to small-size AUVs [SNK16]. MRAS requires the determination of the sensitivity derivative that cannot be obtained for an unknown process (unless several assumptions are made). Adaptive methods have been extensively used following these model-based schemes when a major part of the process model is available. Nevertheless, the limits of model-based adaptive methods have also been well outlined [And05], restricting their application to processes of limited uncertainty.

In Figure 2.3 a block diagram of a model-reference adaptive system (MRAS) is provided, which is composed of an inner ordinary feedback loop enclosing the process with the controller and an outer loop that includes the parameter adjustment mechanism. The adjustment mechanism is obtained either by using gradient methods or by applying stability theory. However, this knowledge of the gradient is not always available.

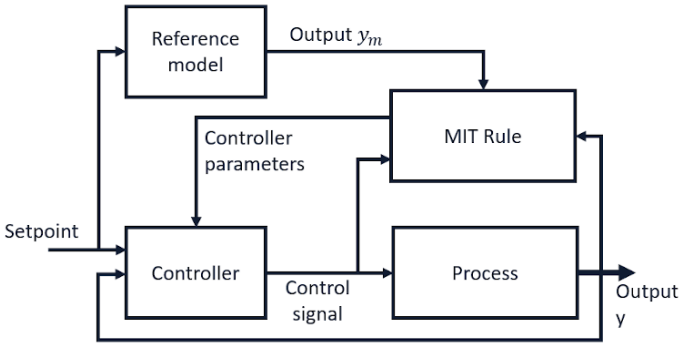


Figure 2.3: *Block diagram of a model-reference adaptive system (MRAS).*

Model-free adaptive control

Adaptive methods that do not rely on any mathematical model of the system are called model-free. These methods aim at describing complex processes from sensory data only, rather than from first principle laws. The first such theories appeared in the French literature in the early 1920's [Leb22]. The concept was then formalized under what is currently known as *Extremum Seeking Control* (ESC). The goal of ESC is to maximize a control objective function $J(\cdot)$ without any knowledge of its extremum $y^\star$. We present now a simple ESC procedure applied to the system (2.1) also known as perturbation-based ESC (PB-ESC). Consider that the control objective $J(\cdot)$ is to maximize the system output y. For simplicity, let's consider a largely simplified case where $J(\cdot)$ is a static paraboloid function of u as illustrated in Figure 2.4 and that there exists a unique $x^\star$ such that $y^\star = h(x^\star)$ is the extremum of the $h(\cdot)$ function. We assume here that (2.1) is SISO, for which we can fairly design an optimal feedback control law as:

$$u = \alpha(x, \theta), \tag{2.4}$$

where $\theta \in \mathbb{R}$ is a scalar parameter. The closed-loop form of system (2.1) considering (2.4) is:

$$\dot{x} = f(x, \alpha(x, \theta)). \tag{2.5}$$

A straightforward first order PB-ESC system can be derived as:

$$\begin{aligned} \dot{x} &= f(x, \alpha(x, \hat{\theta} + a\sin(\omega t))), \\ \dot{\hat{\theta}} &= kh(x)b\sin(\omega t), \end{aligned} \tag{2.6}$$

where (k, a, b, ω) are tuning parameters. As shown in Figure 2.4, this algorithm contains only one integrator and can be summarized as:

1. Inject some sinusoidal signal on the current control input $\hat{u}$.
2. Measure the sinusoidal perturbation on the resulting $J(\cdot)$.
3. Integrate the multiplied signal to the control input $\hat{u}$.
4. Repeat from step 1.

Because of the shape of $J(\cdot)$, the control $\hat{u}$ can either be in phase (when it is on the left of the optimum) or out of phase (when it is on its right). Thus, the integration ensures that we move toward $u^\star$. This simple PB-ESC can be applied to processes that are subject to variations and uncertainties where the function $J(\cdot)$ is not static. The PB-ESC algorithm can track the optimal u^* only if the disturbances and process parameters are changing slowly compared to the sinusoidal probing. It was shown using averaging and perturbation theories that this process can (locally) converge, under some assumption (of local optimality and smoothness of J) toward a neighborhood of u^* [Rot00]. The PB-ESC algorithm is a model-free adaptive control scheme because the parameter θ of the control law u is adjusted solely based on the system feedback.

PB-ESC is essentially a local optimizer and real-life dynamical systems exhibit $J(\cdot)$ functions with multiple peaks and discontinuities. In this case, the nonlinear version of ESC can be used [KW00] which consists mostly of the addition of low-pass and high-pass filters. The convergence analysis of ESC has also been conducted showing that convergence to a global maximum in the presence of multiple local maxima can indeed be achieved with proper tuning of the control parameters [Nes09; Biz20]. ESC algorithms have also been applied to tune PID controllers [KK06; KK05].

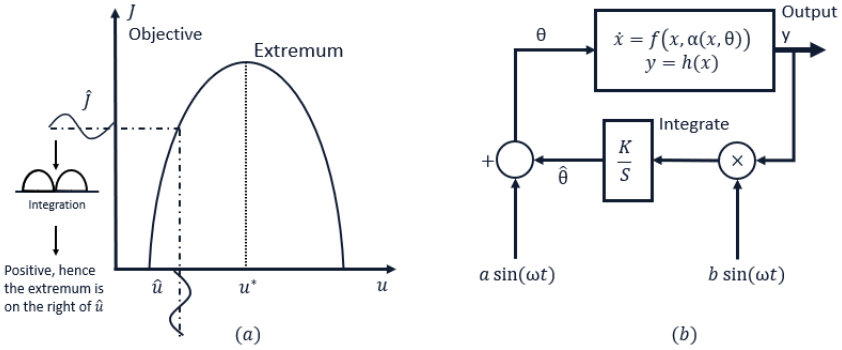


Figure 2.4: *Illustration of the objective function $J(\cdot)$ considered for this example (a) and the associated PB-ESC system (b).*

There exist various other optimization techniques to find extrema (e.g. Hill Climbing, Nelder–Mead method or Simulated annealing). Today, the most commonly used model-free algorithm is Reinforcement Learning [BT96; SB18]. The hypothesis underlying Reinforcement Learning is that, by trying multiple control inputs at random, a controller can eventually (by trial and error) build a predictive model of the system on which it is operating. The optimal control policy is hence obtained through this experimental process that leads to the well-known exploration versus exploitation tradeoff. Some Reinforcement Learning methods use this trial and error scheme not to learn the state-to-optimal actions mapping, but rather to learn the model of the process. This model can then be used for planning future actions. Over the past decades, model-free adaptive control theory has reached a reliable level of maturity with good analysis and understanding of its main properties. Thenceforth, great efforts have been made in order to take advantage of the model-based design, with its stability characteristics, and add to it the advantage of model-free learning, with its fast convergence and robustness to uncertainties. The field of learning-based adaptive control was later developed having this ambition as the goal.

Learning-based adaptive control

Real-world systems are in general nonlinear and their motion equations, parameters and system measurements are affected by uncertainty. A more realistic scheme is to consider that the process model is partially available. In learning-based adaptive controllers, model-free algorithms are used to mitigate this lack of a complete description of the process by finding (*learning*) an approximate representation of the unknown parts, or by fitting (*tuning*) the best control parameters for a target behavior. The dynamics represented in (2.1) can be re-written as the sum of known (f_1) and unknown (f_2) parts of the process:

$$\begin{aligned} \dot{x}(t) &= f_1(t, x, u) + f_2(t, x, p), \\ y(t) &= h(t, x, u), \end{aligned} \tag{2.7}$$

where classical model-based control methods can be used to efficiently control f_1, and f_2 can be approximated by model-free learning algorithms. In other words, learning-based control methods take advantage of the fast convergence and robustness to the uncertainty of learning algorithms to approximate an unknown performance function, while applying model-based control laws to obtain optimal performance of the system. as formalized below, where a state-space model of (2.1) is described in the Brunovsky form as:

$$\begin{cases} \dot{x}_1 = x_2, \\ \dot{x}_2 = f(\cdot) + u, \end{cases} \tag{2.8}$$

where $f(\cdot) = f_1(x(t), u(t), t) + f_2(x(t), p(t), t)$, f_1 is a known function, f_2 is an unknown function, both functions are smooth over the state variables $x = (x_1, x_2)^T$, and u the control signal. The unknown part of the model (f_2) is estimated by a NN as:

$$\hat{f}_2 = \hat{W}^T S(x(t)), \tag{2.9}$$

where $\hat{W} = (\hat{w}_1, \ldots, \hat{w}_N)^T \in \mathbb{R}^N$ is the estimated vector of synapse weights of the neural network node and $S(x) = (s_1(x), \ldots, s_n(x))^T$ is the regressor vector, with $s_i, i = 1, \ldots, N$. Given the reference model:

$$\begin{cases} \dot{x}_{ref_1} = x_{ref_2}, \\ \dot{x}_{ref_2} = f_{ref}(x), \end{cases} \tag{2.10}$$

the function f_{ref} is a known nonlinear smooth function of the desired trajectories $x_{ref} = (x_{ref1}, x_{ref2})^T$. A basic learning-based controller can be defined as:

$$\begin{aligned} & u = -e_1 - c_1 e_2 - \hat{W}^T S(e) + \dot{v}, \\ & \text{with} \\ & e_1 = x_1 - x_{ref1}, \\ & e_2 = x_2 - v, \\ & v = -c_2 e_1 + x_{ref2}, \\ & \dot{v} = -c_2(-c_2 e_1 + e_2) + f_{ref}(x_{ref}),\ (c_1, c_2 > 0), \\ & \dot{\hat{W}} = \Gamma(S(e)e_2 - \sigma\hat{W}),\ (\sigma > 0 \text{ and } \Gamma^T > 0). \end{aligned} \tag{2.11}$$

It can be observed that u is now a function of the Brunovsky form (2.8) (i.e., model-based information), and the remaining part is based on the neural network estimates of f_2 (model-free estimation), thus the name learning-based adaptive controller. Following this postulate, many Extremum-Seeking-based learning-based methods have been proposed in the literature, such as [XB15; HA13]. Other optimization methods such as Genetic algorithms [Dav90; Gol08] and Evolution Strategies [Wie+08; Con+18] have also been proposed in the learning-based framework. More recently, in Neural-Network (NN) learning-based control design, the unknown part of the model can be estimated by a NN, whose weights are obtained using some model-free optimization procedure. Among the various Machine Learning techniques, a prominent candidate for that end is Reinforcement Learning (RL). In the next section, we present the background elements of RL and we will describe how it can be used for AUVs' learning-based adaptive control.

2.2 Reinforcement Learning

Reinforcement Learning (RL) as it is referred to today, is a study, and a problem of intelligence attempting to discover what could drive agents (natural or artificial) to behave intelligently in such a variety of ways. As pictured in Figure 2.5, RL is often framed in the form of an agent that interacts with an environment through actions, and one particular thing that the environment generates is called the reward signal which tells how well the agent is doing in the environment. The objective of RL is simple, that is to make rational decisions that are decisions maximizing a measure of utility, portrayed by the reward. Rational decision-making can be seen, through a particular lens, as the encapsulation of the artificial intelligence problem. Any problem that we would want a machine to do can likely be represented as a rational decision-making problem (like classifying images, controlling a chemical plant, or deciding what movies to recommend on Netflix) and RL provides a framework to design solution methods for this class of problem. For this reason, great efforts were made in scaling RL, by both the academic and industrial communities. RL has now proved its efficiency in many applications in science and engineering with the remarkable advantage that is, it enables agents to maximize their cumulative rewards through online exploration and interactions with unknown (or partially unknown) and uncertain environments, which is regarded as a variant of data-driven adaptive optimal control theory. Nowadays, RL methods are coupled with deep machine learning methods, allowing their use in applications where no solutions were previously available. The goal of this Section 2.2 is to provide an overview of the aforementioned notions that are the building block of modern RL and why it is suitable for the considered control problem. The fundamentals of RL will be presented, and some theorems and proofs might be described later in the document and will be cited throughout when appropriate.

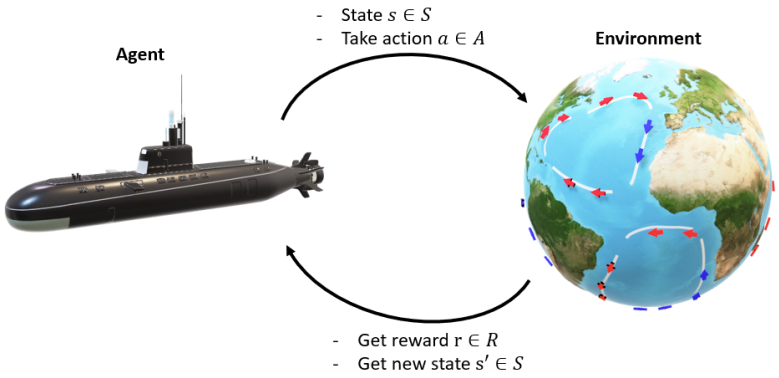


Figure 2.5: *Illustration of an RL process where the agent is interacting with its environment.*

RL is defined as a class of solutions methods for learning how to interact with the environment from raw experience. At the heart of RL, we start with an agent and an environment. The term agent implies some agency, as the agent gets to take action by following its behavior (policy). The agent is said to be able to capture a snapshot of itself (state) within the environment. In order to improve its performance, the agent can learn either from interactions generated by its current policy (On-Policy) or from interactions generated by any policies (Off-Policy). These past interactions would need to be held in some kind of memory unit (Replay Buffer) which we would often appraise to refresh our beliefs. From these past interactions, the agent has to extract and build meaningful understandings (model) of the environment response (value function). Using this knowledge, the agent can then take the best action to take (optimal policy). RL is a framework allowing us to learn the optimal policy in a model-free setting, where the model of the process is partially or fully unknown. For this reason, using RL in the context of underwater robotic applications is a pertinent choice given that underwater vehicles have a limited observation ability of the environment and have to compensate for various disturbing forces that are often not measurable while having limited computational resources.

2.2.1 Key concepts of RL

Let's first define a set of key concepts required for the understanding of RL.

The agent is acting in what we call an environment. The environment can be represented by different means, such as an actual physical space where the agent is evolving, or by something more abstract such as a set of rules. In all cases, how the environment reacts to certain actions is defined by a model, which may be known or not. In RL, the model is a descriptor of the environment that can be used to learn of infer how the environment would react to and provide feedback to the agent. The agent is defined at every time in one of many states of the environment $s \in S$ and chooses to take some actions among a predefined set $a \in A$, whose execution make the agent transit to another state $s' \in S$. Following this transition, the environment generates, in addition to a new state, a reward signal $r \in R$ which transcribes numerically to how good this choice of actions was with respect to the goal of future reward maximization.

The action selection procedure is represented by what is called the policy $\pi(s)$. It is a mapping from the state s to actions a and can then be either deterministic or stochastic. For the goal of reward maximization, each state is associated with a value, denoted as state-value $V(s)$, which assesses the expected amount of future reward accessible from the considered state and by acting accordingly to the policy. In other words, the state-value quantifies how good a state is. In general, both the policy and state-value are what we try to learn in RL.

The interaction between the agent and the environment involves a sequence of actions, states, and rewards observed at the time, $t = 1, 2, \ldots, T$. During the learning phase, the agent is expected to collect knowledge about the environment, build the optimal policy accordingly to its current understanding, and make decisions on which action-making strategy to take so as to efficiently learn the best policy. Now, we can label the state, action, and reward at the timestep t as s_t, a_t, and r_t respectively. In RL, the sequence of interaction is fully described by an episode (also denoted as trial or trajectory) which ends at the terminal state s_T:

$$s_1, a_1, s_1', r_1, s_1', a_2, s_2', r_2, \ldots, s_T. \tag{2.12}$$

The model is a description of the environment, from which we can learn or infer how it will react to the agent's actions. The model consists of two parts: the transition probability function P and the reward function R. A transition is defined as follows: from a state s, the agent takes the actions a_t, transiting to a new state s' and obtaining r. A transition step is therefore framed as the tuple $\langle s, a, s', r \rangle$. The transition probability function P accounts for the probability of transit from state s to state s' after taking action a while generating the reward r. Formally, this function is defined using the symbol $\mathbb{P}$ (for probability distribution) as follows:

$$P(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]. \tag{2.13}$$

The state-transition function can then be defined as a function of $P(s', r|s, a)$:

$$\begin{aligned} P^a_{ss'} &= P(s', r|s, a) \\ &= \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a] \\ &= \sum_{r \in R} P(s', r|s, a). \end{aligned} \tag{2.14}$$

Similarly, the reward function predicts the reward generated by the actions:

$$\begin{aligned} R(s, a) &= \mathbb{E}[R_{t+1}|S_t = s, A_t = a] \\ &= \sum_{r \in R} r \sum_{s' \in S} P(s', r|s, a). \end{aligned} \tag{2.15}$$

Following the above definition of the transition probability function, the policy, which is a mapping from the state s to actions a, is defined as:

$$\begin{aligned} &\pi(s) = a \text{ (Deterministic)} \\ &\pi(s|a) = \mathbb{P}_\pi[A = a, S = s] \text{ (Stochastic).} \end{aligned} \tag{2.16}$$

In order to measure how good it is for an agent to be at a given state, the state-value function estimates the expected future cumulative reward obtained starting from the considered state. This quantity is denoted as the agent return G_t, and is equal to the total sum of discounted future rewards going forward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.17}$$

The parameter γ, known as the discount factor, is incorporated in the return such as $0 < \gamma < 1$. As the discount factor is less than 1, this factor penalizes far future rewards by shrinking their value toward 0. The addition of this parameter has two effects:

- Discounting provides mathematical convenience by bounding the infinite summation. By considering such a design, we do not need to worry about infinite loops in the state-transition graph.
- Depending on the value of γ, we can control how far in the future we look in the resulting state-value function. With a $\gamma \sim 0$, the agent is short-sighted and puts way more emphasis on the short-term in his decision-making. With a $\gamma \sim 1$, the decision-making takes into account very far predictions, which do not have any immediate benefits. The value of the discount factor is thus a parameter to tune depending on the agent, environment, and desired performance.

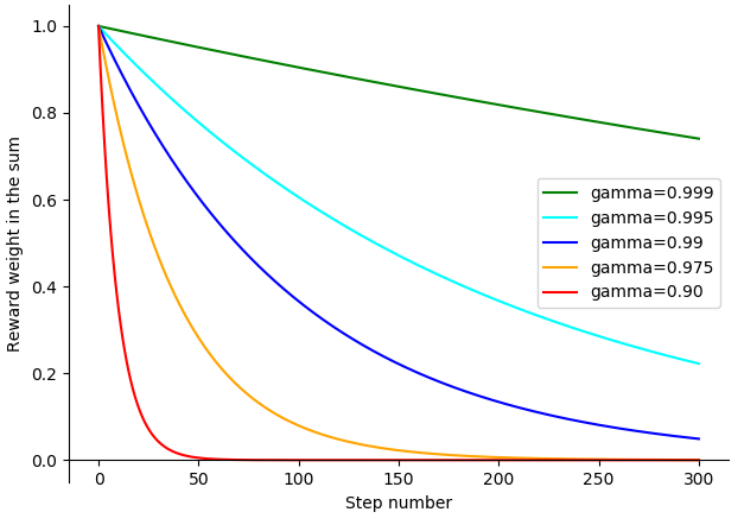


Figure 2.6: *Illustration of the impact of the discount factor on future rewards.*

The discount factor has another objective, which is to reduce the variance of the return cf Eq. (2.17). There is uncertainty about the far future state of the environment because the further we look into the future, the more stochasticity we accumulate and the more variance the return will have. Under these conditions, estimating the return can be very difficult and the choice of discount factor should therefore be influenced by the environment's complexity. In addition, in Figure 2.6 we can see that for each value of γ, after a certain threshold of step number, the generated reward will have little to no impact on the discounted return. For example, with $\gamma = 0.90$ (i.e. red line in Figure 2.6), this threshold is approximately equal to 50. Let's now consider that the length of the episode exceeds this threshold. The resulting actions will then be the best ones for only a portion of the episode. Thus, the agent is biased because what we believe are optimal actions, are mostly optimal only over this reduced period of time, and thus could be sub-optimal (or even far from it) with regards to the entire episode. Potentially, better future states could be accessible from different early actions.

For this reason, the discount factor is a parameter to tune also according to the length of the episode. One designer should ensure that the entire trajectory will have a notable impact on the computation of the return. At the same time, the discount factor should not be too close to 1 so as to make the return estimation feasible. The design of these parameters (i.e. whether or not to set the discount factor based on the episode length or the other way around) is a choice to make based on the environment, the task, and the desired usage of the resulting policy.

The more we look into the future when estimating the Q-Value function (i.e. $\gamma \mapsto 1$), the more stochasticity we accumulate and the more variance the estimate will have. Although a discount factor lower than 1 notably stabilizes the learning process, it should not be too small in order to avoid a short-sighted agent. In our application, the control system of the AUV needs to take into account future states of the process because due to the disturbance, what we believe to be optimal with respect solely to the current timestep (i.e. $\gamma = 0$) can be far from it or even dangerous for the platform in the long term.

The state-value function can now be defined as the expected return starting from the current state:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]. \tag{2.18}$$

The state-value function measures only the quality of a given state. As the goal is to take the best action possible, we would like to include the policy in this evaluation through what we could call an "action-value"

function. The quality of a pair of state and action (s, a) is denoted as the Q-value function ("Q" for quality) and defined as:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{2.19}$$

The Q-value function (2.19) measures how good it is for the agent to follow the current policy starting from the evaluated state. Since we follow the target policy π, we can recover the state-value function using the probability distribution over the possible actions:

$$V_\pi(s) = \sum_{a \in A} Q_\pi(s, a)\pi(a|s). \tag{2.20}$$

Finally, as the goal of RL is to learn the best policy possible, we will be interested in the optimal value of the aforementioned functions. In this document, the optimal value functions will be denoted as $V_*(s)$ and $Q_*(s, a)$ and are defined as:

$$\begin{aligned} V_*(s) &= \max_\pi V_\pi(s), \\ Q_*(s, a) &= \max_\pi Q_\pi(s, a). \end{aligned} \tag{2.21}$$

Accordingly, the optimal policy π_* achieves optimal value functions:

$$\begin{aligned} \pi_* &= \arg\max_\pi V_\pi(s), \\ \pi_* &= \arg\max_\pi Q_\pi(s, a). \end{aligned} \tag{2.22}$$

In the context of adaptive control of AUVs, the RL-based optimal policy (2.22) can consists of estimating at each timestep the best control inputs to apply or the best controller parameters to use for the objective of minimizing the regularization error. This is possible despite the unobservable current disturbance as the State-Value function (2.20) captures and encircles all of the things in which an agent interacts with an environment, including the aforementioned uncertainties. In the following, we will present how we can formalize mathematically this optimization procedure in order to apply it to our AUV application.

2.2.2 Markov decision process

Mathematicians have provided different tools to frame the problem of RL, and historically, the Markov Decision Process (MDP) describes a framework to solve RL. MDPs are discrete-time stochastic control processes and almost all RL problems can be formalized as MDPs (essentially all processes where we can have feedback and thus derive a reward signal). As pictured in Figure 2.7, we can see MDPs as extensions of Markov Chains, the difference being the addition of actions (i.e. choice) and reward (i.e. optimization objective). A stochastic control process is considered an MDP if each state of the process holds the Markov Property. A state S_{t+1} is said to be Markovian if and only if:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]. \tag{2.23}$$

MDPs have been studied for decades, and various solutions methods have been proposed to solve them. In the following, we will introduce the methods that are used in our proposed methods. MDPs are expressed as the tuple $\langle S, A, T, R\rangle$, in which:

- S is the set of possible states;
- A is the set of actions that can be executed by the agent;
- T is the transition function that defines the probability of reaching a successor state $s' \in S$, from the application of action $a \in A$ in a state $s \in S$;
- R is the reward function.

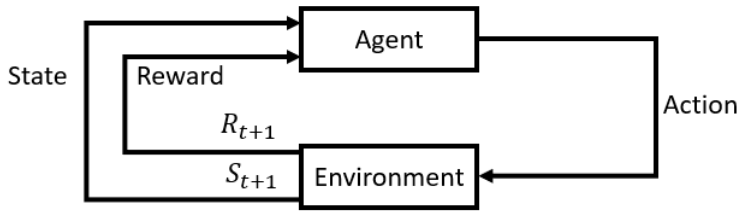


Figure 2.7: *Illustration of the agent-environment interaction in a Markov Decision Process.*

2.2.3 Bellman equations

If an optimization problem can be decomposed into various subproblems, Dynamic Programming (DP) [Ber95; Put94] can be used to solve it. In this case, the Bellman equations [Bel52] can be used as a substitute for the model of the process. It is the fundamental mathematical tool used to solve MDPs. It allows us to decompose a dynamic optimization problem into a sequence of subproblems, for which we can prove that an optimal solution exists. Here, these equations allow us to express the relationship between the value of a state and the value of the successor state. In practice, the Bellman equations allow us to re-write the state-value function as the immediate reward plus the discounted future value function:

$$\begin{aligned} V(s) &= \mathbb{E}[G_t|S_t = s], \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots |S_t = s], \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots)|S_t = s], \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s], \\ &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}|S_t = s)]. \end{aligned} \tag{2.24}$$

Similarly, the Q-value function can be decomposed as:

$$\begin{aligned} Q(s,a) &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s, A_t = a], \\ &= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a\sim\pi} Q(S_{t+1}, a)|S_t = s, A_t = a]. \end{aligned} \tag{2.25}$$

We know that from a given state s, we can choose actions from multiple possibilities determined by the stochastic policy $\pi(a|s)$, and each of these actions is associated with a Q-value. By multiplying the possible actions with the Q-value function and summing them, we can derive an indicator of how good it is to be in that given state, which is also known as the Bellman Expectation equation:

$$V_\pi(s) = \sum_{a\in A} \pi(a|s) Q_\pi(s,a). \tag{2.26}$$

From this equation, we can assess the value of a state by multiplying the possible actions with the action-value function and summing them. Similarly, for a list of possible actions, there is a list of possible next states s' which are associated with a state-value function $V(s')$, a transition probability function $\mathcal{P}^a_{ss'}$ of where the agent could end up based on the actions, and a reward R^a_s generated by taking the action. By summing the reward and the transition probability function associated with the state-value function, we can now derive an indicator of how good it is to take the action, given a state:

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s'\in S} \mathcal{P}^a_{ss'} V_\pi(s'). \tag{2.27}$$

Substituting the recursive Q-value function (2.27) into the recursive state-value function (2.26) we derived what are known as Bellman Expectation equations:

$$\begin{aligned} &V_\pi(s) = \sum_{a\in A} \pi(s|a)(R(s,a) + \gamma \sum_{s'\in S} \mathcal{P}^a_{ss'} V_\pi(s')), \\ &\text{and} \\ &Q\pi(s,a) = R(s,a) + \gamma \sum_{s'\in S} \mathcal{P}^a_{ss'} \sum_{a'\in A} \pi(a'|s') Q_\pi(s',a'). \end{aligned} \tag{2.28}$$

In order to solve an RL problem that is framed as an MDP, we are interested in the optimal values of the Bellman Expectation Equations (2.28) rather than computing the expectation following a policy:

$$\begin{aligned} V_*(s) &= \max_{a\in A} Q_*(s,a), \\ Q_*(s,a) &= R(s,a) + \gamma \sum_{s'\in S} \mathcal{P}^a_{ss'} V_*(s'). \end{aligned} \tag{2.29}$$

Again, by substitution, we can derive what are known as Bellman Optimality Equations:

$$\begin{aligned} &V_*(s) = \max_{a\in A}(R(s,a) + \gamma \sum_{s'\in S} \mathcal{P}^a_{ss'} V_*(s')), \\ &\text{and} \\ &Q_*(s,a) = R(s,a) + \gamma \sum_{s'\in S} \mathcal{P}^a_{ss'} \max_{a'\in A} Q_*(s',a'). \end{aligned} \tag{2.30}$$

The field of RL then consists mainly in designing solution methods that determine the Bellman Optimal Equations (2.30). In Section 2.2.4, we proposed to classify solutions methods based on the nature of the interactions with the environment, generating the rewards that are mandatory to estimate the above functions (2.30). We propose to classify methods in such a manner because we are mainly interested in mobile robotic applications. In order to interact with their environment, these types of agents have to directly explore their surroundings. Depending on the system and environment, exhaustive interactions (such as required in general by RL methods to be efficient) might not be possible or efficient because: the policy (especially at an early stage of training) can be dangerous for the robotic platform because of hardware vulnerability; the operating cost can be too high to consider thorough exploration of the environment due to time restrictions; the process uncertainty can be so large that the Markov property does not hold anymore. With the proposed classification, we can easily identify which class of solution methods to favor according to the primary knowledge of the robot and operating conditions.

To summarize, the Bellman equations can be used as a substitute for the model of MDPs and RL is a technique to solve a set of Bellman equations. When determined, the adaptive control problem of AUVs which can be framed as an MDP can therefore be solved using these functions despite the model of the associated MDP not being a priori known. Accordingly, RL allows us to obtain a solution to the adaptive control problem of AUV that is not possible when using model-based adaptive control theory solely. Next, we present a classification of solution methods for estimating these functions.

2.2.4 Classification of solution methods

Reinforcement Learning is a very ambitious problem definition as it is trying to capture and encircle all of the things in which an agent interacts with an environment. Maybe the complexity of the environment is so great that we can not even imagine how to build a system that would understand how to take action just from a stream of observations. To even start thinking about how to solve such a hard problem, the first step we can take as human beings is to decompose (when it is possible) that big hard problem into pieces that work together to solve that hard problem. By taking a look at the decomposition that might be inside the agent's head, we can ask what form that decomposition could take. This decomposition commonly consists of some building blocks and solution methods in RL are different choices of using them, and some of the most common pieces that people use when they are trying to put solution methods together are:

- is there a *model* in the system, that is something that is explicitly trying to predict what will happen in the environment?
- whether or not that solution has a *value* function, that means is it trying to predict, explicitly, how much reward it will get in the future?
- or does it have a representation of a *policy*, that means something that is deciding how to pick actions, is the decision-making process explicitly represented?

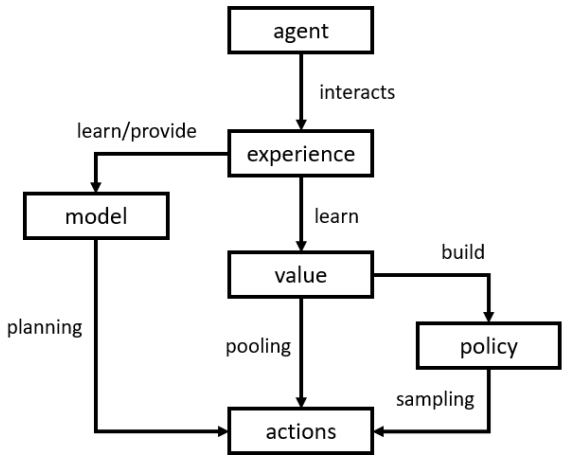


Figure 2.8: *Illustration of RL methods based on the nature of the decision-making process: in model-based methods, the actions are the result of deterministic planning, in value-based methods the actions are pooled over the entire set of possible actions, and in policy gradient methods the actions are sampled from a probability density function.*

These three pieces are the most common building blocks, and as illustrated in Figure 2.8 with the three classes of RL methods and their attribute, solution methods in RL are different choices of whether or not to use them. The fundamental idea of this decomposition is to ask how could we solve problems, that can be decomposed into smaller problems, where we are trying to figure out how to take actions just from this stream of observation. The first step of this decomposition is to say that we have to learn, the system has to learn for itself. Learning is required because it is good at achieving good performance in large and complex environments.

This step gives rise to all the other pieces because now we might ask what should we be learning, and what learning does even mean? In this context, learning might mean that we are trying to update the parameters of some system which is then the thing that actually picks the actions. Those parameters could be representing anything, a Value function, a model, or a policy. In that sense, there is a lot of commonality between these building blocks in whatever is being represented there, the thing which is being learned with the ultimate goal of maximizing the reward. Solution methods in reinforcement learning are inherently different choices of whether or not to use them. In this thesis, we use RL methods that exploit each of these building blocks. For this reason, a concise presentation of these concepts is provided in Sections 2.2.4, 2.2.4 and with an extended focus in Section 2.2.4 on the class of solution methods denoted as Policy Gradient that is the main component of the contributions of this thesis.

Model-based reinforcement learning

When the model $P(s', r|s, a)$, see Eq.(2.13), is fully known, the RL problem can be framed as a planning problem. In model-based methods, the rewards are not the results of the interaction with the environment but they are given by the model that represents the dynamics of the associated MDP. This can be seen as a deterministic optimization procedure, where we know for each action where the agent will transit in the environment and the associated state-value, and the most common approach for this is called **Dynamic Programming** (DP) [Ber95; Put94]. This method can be used when the model is known to solve either: a prediction problem (i.e. policy evaluation); or a control problem (i.e. policy improvement). The *policy evaluation* consists in assessing how good a policy π is, given an MDP. The idea is to start with an initial state-value function V_1 with a value of 0 and then to update it to V_2 using the Bellman backup (2.24). This process is repeated many times, until convergence to V_π, the state-value function associated with the evaluated policy:

$$V_1 \to V_2 \to \cdots \to V_\pi. \tag{2.31}$$

The $V(s)$ function is updated by using synchronous backups and all states at every step with:

$$V_{t+1}(s) = \mathbb{E}_\pi[r + \gamma V_t(s')|S_t = s] = \sum_{a\in A} \pi(s|a) \sum_{s'\in S, r\in R} P(s', r|s, a)(r + \gamma V_\pi(s')). \tag{2.32}$$

This is illustrated in Figure 2.9 with the backup diagram of a DP procedure where from a state s_t, all following states are considered in the State-Value estimate Eq. (2.32). The *policy improvement* consists in generating a better policy π'. In fact, with this evaluation (2.32), we have assessed the performance of a given policy π but have not found the best ones for our environment. To improve the policy, we can act greedily with respect to the determined state-value function. This is possible using a one-step look ahead to determine the action which maximizes the Q-value function: $\pi'(s) = \arg\max_{a\in A} Q_\pi(s, a)$. By acting greedily we obtain a better policy $\pi' > \pi$:

$$Q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s, A_t = a] = \sum_{s'\in S, r\in R} P(s', r|s, a)(r + \gamma V_\pi(s')). \tag{2.33}$$

In order to obtain the optimal policy, one can intuitively understand that we will need to combine both policy evaluation and policy improvement. This is exactly the goal of what is denoted as **Generalized Policy Iteration** (GPI) algorithm which iteratively alternates between these tasks:

$$\pi_0 \xrightarrow{evaluation} V_{\pi_0} \xrightarrow{improvement} \pi_1 \xrightarrow{evaluation} V_{\pi_1} \ldots \xrightarrow{improvement} \pi_* \xrightarrow{evaluation} V_{\pi_*} \tag{2.34}$$

With GPI, the state-value function is approximated repeatedly so as to be as close as possible to the true value of the current policy, and at the same time, the policy is improved repeatedly to reach optimality. We can ensure that this iterative process converges toward the optimal policy, in other words, that the value of the improved policy V'_π is better than the previous one V_π because:

$$Q_\pi(s, \pi'(s)) = Q_\pi(s, \arg\max Q_\pi(s, a)) = \max_{a\in A} Q_\pi(s, a) \geq Q_\pi(s, \pi(s)) = V_\pi. \tag{2.35}$$

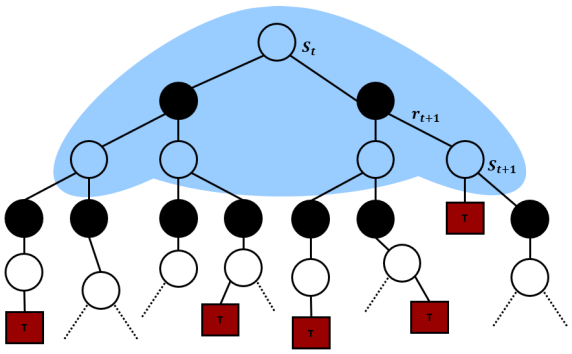


Figure 2.9: *Backup diagram of DP-based method for the state-value function [SB18].*

Model-based RL can be applied essentially when we have full knowledge of the considered process (e.g. when we can write down as equations the exact evolution of the process dynamics at every timestep). In this case, the model of the MDP (2.13) is known and can be used within the GPI method. However, in our application, the AUV is facing sea current disturbances that are not measured. The effect of this disturbing force will also vary depending on the AUV's relative orientation to the sea current which we don't know how to model. The control of AUVs is also associated with various uncertainties that we are yet able to model, including water temperature and salinity affecting the vehicle buoyancy, thruster, and propeller power loss which can make the vehicle underactuated or body wrench disturbances. Model-based RL is thus not applicable to our application because it is impossible here to determine beforehand the MDP's transition probability function (2.13) given the aforementioned uncertainties.

Value-based reinforcement learning

AUVs have to evolve in highly uncertain environments where we are still not able to model the undergoing natural phenomena. The model $P(s', r|s, a)$ of the MDP is therefore unknown and model-based RL can not be used. In this context, we can still solve the RL problem by using Bellman value functions as a substitute for the model without the need to model the environment dynamics. In fact, the state-value function allows us to estimate the agent return as the discounted cumulative reward $V(s) = \mathbb{E}[G_t|S_t = s]$. The most common approaches for this are called **Monte-Carlo** (MC) methods [Has70; RC04; Moh+20] invented by Stanislaw Ulman in the early 1940s. The MC methods are based on the following idea: apply repeated random sampling to obtain numerical results for difficult or otherwise impossible problems. The concept is to consider the value of a probabilistic event as the mean value observed over a great number of repeated events. It allows us to learn from raw experience without the need for modeling the environment dynamics and instead it computes the observed mean return as an approximation of the expected return G_t. MC methods need to learn from complete episodes (i.e. full sequences of $S_1, A_1, R_1 \ldots, S_T$) in order to compute $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+1}$. This is illustrated in Figure 2.10 with the backup diagram of MC where starting from a state s_t, all the future states until the terminal state T are considered in the return estimate. The empirical return is then estimated as:

$$V(s) = \frac{\sum_{t=1}^{T} \mathbb{1}[S_t = s] G_t}{\sum_{t=1}^{T} \mathbb{1}[S_t = s]}, \tag{2.36}$$

where $\mathbb{1}[S_t = s]$ is the binary characteristic function of a subset of a set. This approximation procedure can be easily extended to the Q-value function as:

$$Q(s, a) = \frac{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a] G_t}{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a]}. \tag{2.37}$$

The most common MC-based method (2.10) is denoted as Markov Chain Monte-Carlo (MCMC) and can be summarized as follows:

1. Improve the policy greedily with respect to the current Q-value function: $\pi(s) = \arg\max_{a \in A} Q(s, a)$.
2. Generate a new episode (until termination) with the new policy π (using ϵ-greedy strategy).
3. Estimate the Q-value function using the samples from the new episode with

$$q_\pi(s, a) = \frac{\sum_{t=1}^{T} (\mathbb{1}[S_t = s, A_t = a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1})}{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a]}. \tag{2.38}$$

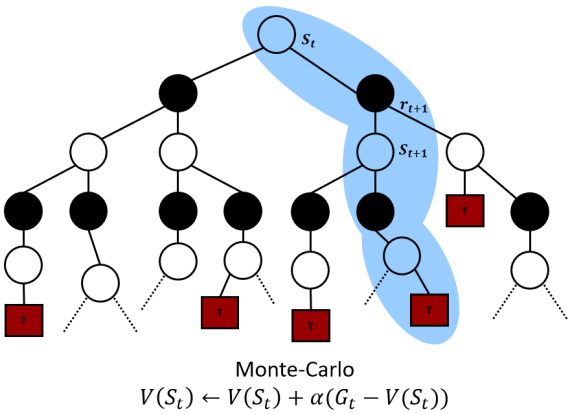


Figure 2.10: *Backup diagram of MC-based methods for the state-value function [SB18].*

Therefore, the MCMC method (similar to GPI) iterates between evaluation and improvement:

$$\pi_0 \xrightarrow[Q\sim q_\pi]{evaluation} Q_{\pi_0} \xrightarrow[\pi\sim greedy(Q)]{improvement} \pi_1 \xrightarrow[Q\sim q_\pi]{evaluation} Q_{\pi_1} \dots \xrightarrow[\pi\sim greedy(Q)]{improvement} \pi_* \xrightarrow[Q\sim q_\pi]{evaluation} Q_{\pi_*}. \tag{2.39}$$

The key difference between DP and MC methods is that we are changing how we are evaluating the policy and estimating the $V(\cdot)$ or $Q(\cdot)$ function. As illustrated in Figure 2.9, the sampling return from DP is different from the Bellman equations because we just use the look-ahead to the next state s_{t+1}, and use it to update our value estimate of the current state s_t. In other words, we update our value estimate only for visited states. Contrary to DP, we don't need to know the states ahead of time, we can just discover them as we interact with the environment, which makes MC much more applicable when the action space dimension is large.

MC methods learn from complete episodes, they can only be applied to episode MDPs. In addition, since we are averaging the value estimate over the full episode, MC methods are often subject to high variance. This is due to the fact that samples collected within the same episode are highly correlated, and updating our estimate using such samples tends to concentrate our updates to specific parts of our estimate, driving us away from the real value of the Q-value function. The real value of a very good action along the episode will be reduced when averaged and similarly, the value of poor action will be enhanced. For this reason, MC-based methods tend to require a large number of iterations before converging to a satisfying solution.

When using MC-based methods such as MCMC, in order to improve the policy the agent needs to visit the exact same states multiple times. We need to interact with the environment a lot. This is not possible in our considered application. Despite having access to the simulation of underwater vehicles and environments, the amount of simulated data remains notably small given the considered continuous action and state spaces that are by definition untrackable. Thus, it is impossible to perform an exhaustive exploration of the underlying spaces or to go back to promising states, which results in the failure of MC-based methods.

To reduce the variance of MC-based Value-based methods, we would like to update our value function using uncorrelated samples. This is exactly the purpose of **Temporal Difference** (TD) learning [Sut+09; SM11; SMW16]. The TD learning methods are model-free methods similar to MC-based methods, but with the advantage of being able to learn the value function from incomplete episodes. This is possible by performing the **bootstrapping** trick which consists in updating *targets* values with regards to existing estimates rather than exclusively relying on actual rewards and complete returns (i.e. MC-based methods). In other words, with TD learning we update a guess toward a guess. The principal idea in TD learning is to update the state-value function towards an estimated discounted return $V(s) = R_{t+1} + \gamma V(S_{t+1})$ (known as TD target). This is illustrated in Figure 2.11 with the backup diagram of TD where from a state s_t, only the next state is considered in the estimation of the return. The TD learning methods are associated with a number of parameters: Gamma (γ) is the discount factor previously introduced in Eq. (2.17); Lambda ($0 < \lambda < 1$) is the credit assignment variable controlling how deep we look into the Markov Chain; and Alpha (α) is the learning rate, controlling how much of the error we accept and adjust our estimate towards.

TD learning methods can further be divided into two groups: on-policy and off-policy. The first one refers to methods where the policy is improved using samples generated by the same policy while off-policy methods use samples generated by any policies to improve the current one. The initial on-policy method to appear in the literature is called **SARSA** (which expands to State, Action, Reward, State, Action), an iterative process similar to GPI. With SARSA, the Q-value function is updated using bootstrapping (i.e. no need for episode termination) and samples are generated by the same policy according to the TD error:

$$Q(S_t, A_t) \xleftarrow[SARSA]{On-policy} Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)). \tag{2.40}$$

The SARSA algorithm can be summarized as follow:

1. Initialize $t = 0$ and set an initial state S_0.
2. for $t = 0, \dots, T$:
   (a) From S_t choose $A_t = \arg\max_{a \in A} Q(S_t, a)$ (using ϵ-greedy exploration strategy in general).
   (b) After applying action A_t, we observed the reward R_{t+1} and transit to the next state S_{t+1}.
   (c) From S_{t+1} pick action similarly to step (a): $A_{t+1} = \arg\max_{a \in A} Q(S_{t+1}, a)$.
   (d) Update the Q-value function as:
   $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$.
3. Repeat from step 2.

With SARSA, for each timestep, we choose the next action according to the current policy. If we always take action according to the same policy, there will be no performance improvement possible (i.e. we are purely doing exploitation). Therefore, the capacity of SARSA to improve the agent return lies in the exploration ability of the policy used to generate the samples. In practice, the ϵ-greedy strategy is used, allowing the agent to take random actions along the trajectory. In addition, as we update our Q-value estimate, the resulting new policy might produce different action a_t'' from that same state S_t. For this reason, and because the efficiency of the ϵ-greedy strategy drastically reduces against continuous state and action spaces, SARSA requires an enormous amount of iteration to converge as it needs to visit each possible state multiple times to improve the policy. To strengthen the sample efficiency of this type of approach, we would want to use samples from any policies to update our Q-value estimate. This is exactly the objective of **Q-Learning** which update the Q-value function by assuming the use of the optimal policy. This is known as off-policy TD learning and we describe this process in Section 2.2.4. Within one episode Q-Learning works as:

1. Initialize $t = 0$ and set initial state S_0
2. At timestep t, we pick action according to $A_t = \arg\max_{a \in A} Q(S_t, a)$, with epsilon-greedy strategy commonly used.
3. Execute A_t, then receive reward R_{t+1} and transit to next state S_{t+1}.
4. Update the Q-Value function as:$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t))$.
5. Set $t = t + 1$ and repeat from step 2.

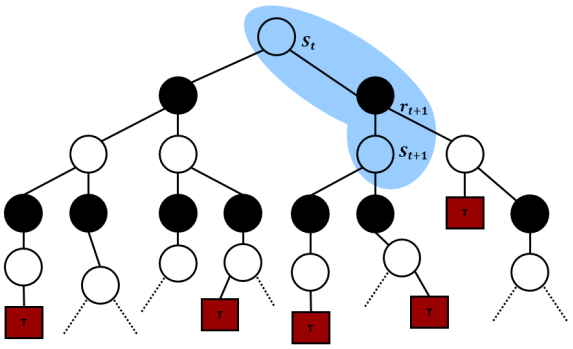
Because we use bootstrapping in TD learning, the difference among the solution methods becomes a tradeoff between bias and variance, which has to be made according to the characteristics of the MDP:

- The return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ (from MCMC) is an unbiased estimate of $V_\pi(S_t)$.
- The TRUE TD target $R_{t+1} + \gamma V_\pi(S_{t+1})$ is also an unbiased estimate of $V_\pi(S_t)$ (but we don't have a heuristic to tell us the true value...).
- The bootstrap TD target $R_{t+1} + \gamma V(S_{t+1})$ is a biased estimate of $V_\pi(S_t)$ (...so we can only substitute in our best estimate so far).
- Using bootstrap, we introduce bias in our estimate and in our target because of $\gamma V(S_{t+1}) \neq \gamma V_\pi(S_{t+1})$.
- TD target is much lower variance than the return because:
  - the return depends on many random actions, transitions, and rewards,
  - while the TD target depends on one random action, transition, and reward.

We have been able to see that the MC-based return is unbiased while the TD target is biased. This means that the MC-based asymptotic prediction error $\overline{VE}(W_{MC})$ is typically smaller. Nevertheless, the TD learning asymptotic prediction error $\overline{VE}(W_{TD})$ can be bounded as follows:

$$\overline{VE}(W_{TD}) \leq \frac{1}{1-\gamma}\overline{VE}(W_{MC}) = \frac{1}{1-\gamma}\min_W \overline{VE}(W) \tag{2.41}$$

where γ is the discount factor introduced in Section 2.2.1. For example, if $\gamma = 0.90$ it roughly corresponds to $\overline{VE}(W_{MC})$ being 10 times smaller than $\overline{VE}(W_{TD})$, but at most. This, in addition to the fact that TD learning converges faster, is why TD learning is leading the field of RL. A value of $\gamma = 0.99$ is traditionally used to ensure an asymptotic error at most 1% bigger than the MC-based one, despite the TD estimate being biased.



Temporal-Difference

$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(s))$

Figure 2.11: *Backup diagram of TD learning methods for the state-value function [SB18].*

The methods described so far allow us to build an estimate of the Q-Value function. The actions are then determined either by planning or by pooling. These methods are advantageous in the case of processes with limited uncertainties such as chess where: we know exactly at each timestep the state of the board and the set of possible moves of the opponent, and we know that new pieces can not appear on the board and the result of each individual move. On the other hand, a little change in the environment that was not taken into account in the Q-Value function estimate will result in the total failure of these methods. In fact, these methods are not satisfying when facing process variation because as they only rely on the Q-Value function estimate for decision-making, the resulting solution is valid only on the exact same set of states. This is never guaranteed in our AUV application. The vehicle is evolving in a dynamic environment where it is facing unobservable process variation and the state and action spaces are continuous. For these reasons, we would rather not rely only on the Q-Value estimate in our decision-making (as it will never be able to encompass all of these uncertainties) but instead learn in addition the behavior policy directly. This class of solution methods is denoted as Policy Gradient. The learned policy $\pi_\theta(a_t|s_t)$ estimates the best action to take given a state. If the policy is learned, despite not being able to fully explore the state of space, the resulting actions will encompass the dynamics of the environment because it is represented by a parameterized function. Facing unseen states, the learned policy function would estimate meaningful actions (as the policy function estimate would be defined on a continuous domain and uncorrelated states are still similar to some extent given that they are still associated with the same physical process). On the other hand, with model-based and value-based methods, in this case, the resulting actions could essentially be meaningless because with these techniques we only learn to fit one particular environment, and any change in the environment will not be reflected in the model. In other words, with model-based and value-based methods we can not recover from unseen process variation because they are not encompassed in the Q-Value estimation. We can expect that, against unseen states, the resulting actions to be at best random. The decision-making obtained when learning the policy explicitly is less sensitive to process variation because since we sample the actions from a policy directly, the resulting solution maintains a physical meaning which makes it most likely to be better than the one sampled from the Q-Value estimate (i.e. if we learning a policy to control an AUV against a sea current that always comes from one direction when facing another current, the policy is more likely to estimate a proper action compare to model-based and value-based methods because there, change in the current disturbance are totally not taken into consideration in the action sampling). For this reason, Policy Gradient methods have been privileged in this thesis.

Policy Gradient

The class of solutions methods presented so far aim to compute (with model-based methods) or learn (in the case of value-based methods) the Q-value function and then select actions accordingly. Instead, policy gradient methods target modeling and optimizing the policy behavior π directly (i.e. the decision-making). The policy is traditionally [Sut+99] represented by a parameterized function with respect to θ, $\pi_\theta(a|s)$. The value of the reward function $J(\theta)$ depends on this policy and thus we can use various algorithms to optimize θ which achieves the best reward. The reward function is defined as the expected return and the parameters θ are optimized with the goal of maximizing the reward function. In discrete space, it is defined with S_1 the initial state as $J(\theta) = V_{\pi_\theta}(S_1) = \mathbb{E}_{\pi_\theta}[V_1]$, or in continuous space as:

$$J(\theta) = \sum_{s \in S} d^{\pi_\theta}(s) V^{\pi_\theta}(s) = \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} \pi_\theta(a|s) Q^{\pi_\theta}(s, a), \tag{2.42}$$

where $d^{\pi_\theta}(s)$ is the stationary distribution of Markov chain for π_θ. For recall, the stationary distribution theorem (sometimes called the Fundamental Theorem of Markov Chains) states that for a very long random walk along within a Markov chain, the probability to end up at some vertex v is independent of where we started the random walk [RCB18]. All of these probabilities in sum are called the *stationary distribution* of the random walk and are unambiguously determined by the Markov chain. Policy Gradient approaches consist in moving the parameter, θ, of the current behavior policy π toward the direction suggested by the gradient of the reward function $\nabla_\theta J(\theta)$ to find the best θ that maximizes the return. However, computing the gradient of (2.42) is not straightforward. The reward function depends on both the current policy (that is directly determined by π_θ) and the stationary distribution of states following the target policy (that is indirectly determined by π_θ). Given that most of the time the model (with respect to model-based RL theory) is unknown, it is difficult to estimate the effect on the state distribution by a policy update. The Policy Gradient theorem [SB18] provides an effective reformulation of (2.42) as:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} Q^{\pi_\theta}(s, a) \pi_\theta(a|s) = \propto \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} Q^{\pi_\theta}(s, a) \nabla_\theta \pi_\theta(a|s). \tag{2.43}$$

Using gradient ascent, we can find the best parameters θ that produce the highest return. The proof of the Policy Gradient theorem (2.43) is provided in the appendix. The policy gradient lays the theoretical foundation of many policy gradient methods as the resulting gradient (2.43) has no bias, but high variance. With this formulation, the parameters θ are updated so as to increase the probability to take the evaluated actions proportionally to their associated Q-value function. This update essentially makes good trajectories (as measured by the Q-value function) more likely and bad ones less likely. However, this is not always guaranteed. Let's imagine that all rewards are large positive numbers, so maybe the best trajectory is associated with a value of 1 million and ten, and the worst trajectory is associated with a value of 1 million and one. With this vanilla formulation (2.43), we will not have the aforementioned update behavior. We will actually have all trajectories becoming more likely as illustrated in Figure 2.12, where the probability to take each action is increased (proportionally to their associated Q-Value) despite action 1 being the best one. This is not the desired policy update behavior and in practice when using this formulation (2.43), it will not work.

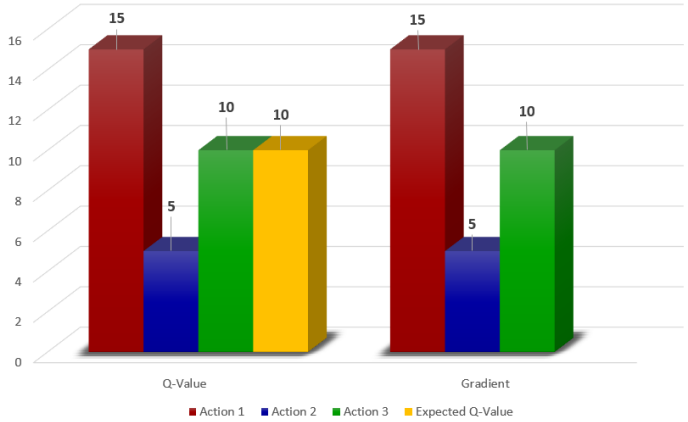


Figure 2.12: *Illustration of the update behavior obtained with the vanilla gradient of the reward function. The values depicted here were chosen only to illustrate that the resulting gradient is proportional to the Q-Value and does not hold any mathematical accuracy.*

For this reason, the development of policy gradient methods was first focused on reducing this variance while keeping the bias unchanged. Intuitively, we would want to make trajectories more likely if they are better than the average one and make them less likely otherwise. This is equivalent to taking our rewards and subtracting the average reward (called *baseline* and noted b when subtracted) from them, which will result in exactly the aforementioned update behavior. This procedure is actually very common in statics and is possible only if the expectation of the baseline is null. In fact, we know that for a given sample-based estimator, if we can subtract off something holding zero in expectation, then the resulting estimate will have less variability between groups of samples because we have subtracted a baseline that we know is zero, but might not be zero for a finite number of samples. We can easily prove that subtracting a baseline is valid as follows:

$$\begin{aligned}
\mathbb{E}[\nabla_\theta \log \pi_\theta(\tau) b] &= \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) b d\tau, \\
&= \int \nabla_\theta \pi_\theta(\tau) b d\tau \rightarrow \text{using the identity: } \pi_\theta(\tau) = \nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau) \\
&= b \nabla_\theta \int \pi_\theta(\tau) d\tau \\
&= b \nabla_\theta 1 \\
&= 0.
\end{aligned} \tag{2.44}$$

This demonstration (2.44) shows that subtracting a baseline is unbiased in expectation. By doing this, the resulting estimator is still correct and in fact will result in a more accurate policy gradient. By using causality (i.e. sum rewards only from the current timestep until the end) and a baseline, these tricks made policy gradient go from an algorithm that never works to an algorithm that sometimes works. In our case, subtracting the average reward will reduce the variance of the estimator of the reward function gradient. This means that for the same number of samples, we will get a more accurate estimate. In practice, the common baseline is the state-value function $V^\pi = \sum_{a \in A} Q^\pi(s,a)\pi(a|s)$. By doing this, the parameters are updated proportionally to their associated difference between the Q-value and state-value functions, which is denoted in the field of RL as *Advantage* function $A(s,a) = Q(s,a) - V(s)$. The gradient of the reward functions becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[(Q^\pi(s,a) - V(s)) \nabla_\theta \log \pi_\theta(a|s) \right] = \mathbb{E}_\pi \left[A(s,a) \nabla_\theta \log \pi_\theta(a|s) \right]. \tag{2.45}$$

This formulation (2.45) is the principal component of the majority of policy gradient methods given that estimating the Q-value function implies that the state-value function is available (i.e. it can be used as a baseline in every case). The value of the Advantage function is positive if the evaluated Q-value is higher than the expected one, otherwise, it is negative. As illustrated in Figure 2.13 with the new associated gradient, the trajectories that are better than the average will be made more likely which is exactly the desired update behavior described earlier.

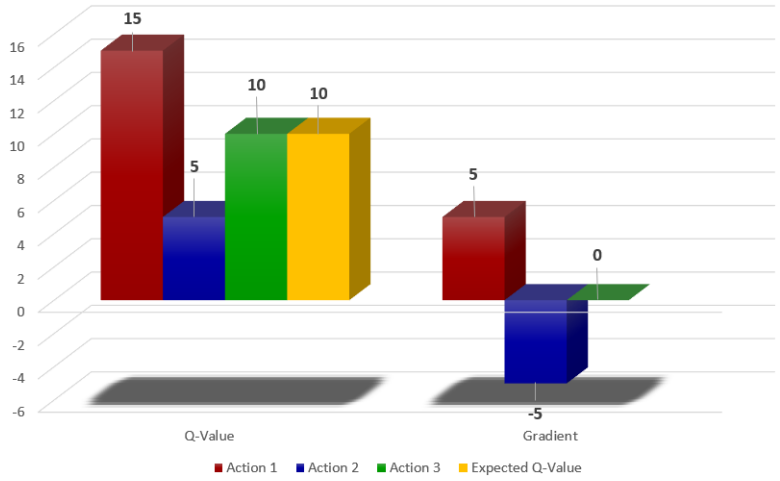


Figure 2.13: *Illustration of the update behavior obtained when using the state-value function as a baseline to reduce the estimator variance. Again, the values used here were chosen only so as to illustrate that when using the Advantage function, the resulting policy gradient update is much more effective.*

The first policy gradient method to appear in the literature is called **REINFORCE** and exploits this baseline trick to *reinforce* the good actions and push down the probabilities of lower actions until we reach the optimal policy. The original REINFORCE algorithm (also known as Monte-Carlo policy gradient) [Wil04] relies on an estimated return by Monte-Carlo methods using episodes samples to update the policy parameters θ. In REINFORCE, a trajectory will be defined as a sequence $\tau = State, Action, Rewards, \dots S_T$. A trajectory is more flexible than an episode because we do not have a length restriction, meaning that the algorithm search for optimal policies for both episodic and continuing tasks. The return G_t of a trajectory is measured again as the sum of discounted future reward (2.17). The algorithm can then be summarized as follows:

1. Initialize θ at random.
2. Use π with parameters θ to generate a trajectory $\tau = S_1, A_1, R_2, S_2, A_2, \dots, S_T$.
3. For $t = 1, 2, \dots, T$:
   (a) Estimate the return G_t.
   (b) Update the parameters as: $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla\pi(A_t|S_t, \theta)$.

The REINFORCE algorithm only learns the parameters θ and computes the expected return directly from the generated trajectory (thus the term *Monte-Carlo* policy gradient). In order to reduce the resulting variance of this Monte-Carlo return estimate, we could instead also learn the value function using TD learning. When the policy and the value function are both learned, the resulting class of solutions methods is named **Actor-Critic** algorithms, and this is the principal RL method used in this thesis because they allow us to use TD learning (which we show is faster to learn to compare to MC and does not need the MDP model contrary to DP) and to learn a policy as well as the value functions (which is more robust to process variations as discussed at the end of Section 2.2.4. As illustrated in Figure 2.14, the Actor-Critic architecture is composed of two components:

- A Critic aims at learning the parameters w of a value function estimate (depending on the algorithm it can be the state-value or the Q-value function).
- An Actor whose parameters are optimized in the direction suggested by the Critic represented as the policy π_θ.

The Actor-Critic algorithm can then be summarized as:

1. Initialize s, w, θ at random
2. Sample $a \sim \pi(a|s;\theta)$
3. For $t = 1, \dots, T$:
   (a) Sample reward $r_t \sim R(s,a)$ and the next state $s' \sim P(s'|s,a)$.
   (b) Sample the next action performed $a' \sim \pi(s', a'; \theta)$.
   (c) Update the policy parameters as: $\theta \leftarrow \theta + \alpha_\theta Q(s,a;w)\nabla_\theta \log\pi(a|s;\theta)$.
   (d) Compute the TD error at timestep t as: $G_{t:t+1} = r_t + \gamma Q(s', a'; w) - Q(s,a;w)$.
   (e) Update the parameters w according to this TD error: $w \leftarrow w + \alpha_w G_{t:t+1}\nabla_w Q(s,a;w)$.
   (f) Move to the next transition $a \leftarrow a'$ and $s \leftarrow s'$.

where α_w, α_θ are learning rates for the policy and value function updates respectively.

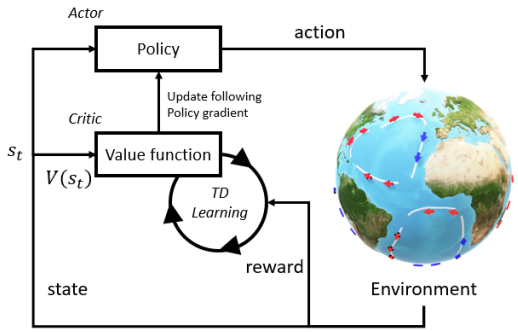


Figure 2.14: *In the Actor-Critic architecture, the value function is learned in addition to the policy. This greatly reduces the variance in the policy gradient update*

Both REINFORCE and the previous Actor-Critic methods are on-policy because the training samples are generated by the same policy that we try to optimize for. This means that we must collect additional samples, by actually running the policy in the environment every time we modify the policy parameters. In fact, as illustrated in the REINFORCE algorithm above, the gradient is an expected value under $\pi_\theta(\tau)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]. \tag{2.46}$$

This causes troubles because that means that every time has to compute the gradient, we have to sample from $\pi_\theta(\tau)$. At every step of training, we have to generate new samples and throw out the old samples. Because of that, the exploration ability is low and a large number of iterations is required for convergence. We would prefer to derive an off-policy approach where:

1. we do not require full trajectories to update the policy, allowing us to reuse any past episodes;
2. the sample collection can follow any policy, different from the target policy, improving exploration.

Now let's derive an off-policy formulation. The policy that is used to interact with the environment is a known policy, denoted as $\beta(a|s)$. The reward function is then defined as the sum of rewards over the state distribution defined by that policy:

$$J(\theta) = \sum_{s\in\mathcal{S}} d^\beta(s) \sum_{a\in\mathcal{A}} Q^\pi(s,a)\pi_\theta(a|s) = \mathbb{E}_{s\sim d^\beta}\big[\sum_{a\in\mathcal{A}} Q^\pi(s,a)\pi_\theta(a|s)\big] \tag{2.47}$$

where $d^\beta(s)$ is the stationary distribution of the policy β, $d^\beta(s) = \lim_{t\to\infty} P(S_t = s|S_0,\beta)$, and Q^π is the Q-Value function estimated with regard to the target policy π (different from β). To achieve the off-policy goal, we essentially want to use samples from any policy to estimate the gradient of the target policy. This is possible using Importance Sampling (IS). It is a method to estimate an expected value of some function under a distribution, given only samples from a different distribution. Let's say we want to estimate the expected value of $f(x)$ under a distribution $p(x)$. According to IS, we can write:

$$\begin{aligned}
\mathbb{E}_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\
&= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\
&= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\
&= \mathbb{E}_{x\sim q(x)}[\frac{p(x)}{q(x)}f(x)].
\end{aligned} \tag{2.48}$$

The IS formulation Eq. (2.48) means that if we have samples from $q(x)$, and we would like to estimate an expectation under $p(x)$, we just have to multiply the samples by $p(x)/q(x)$ which is called an importance weight. Samples that are more likely under $p(x)$ and less likely under $q(x)$ are deemed as being more important, thus this importance weight is a correction allowing us to get the expectation of the desired distribution. We can now apply this idea to estimate expectation under $\pi_\theta(\tau)$ using only samples from any different policy $\bar{\pi}(\tau)$:

$$\begin{aligned}
&J(\theta) = \mathbb{E}_{\tau\sim\bar{\pi}(\tau)}[\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)}r(\tau)] \\
&\pi_\theta(\tau) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t) \\
&\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} = \frac{p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1|s_t,a_t})}{p(s_1)\prod_{t=1}^{T}\pi(a_t|s_t)p(s_{t+1|s_t,a_t})} = \frac{\prod_{t=1}^{T}\pi_\theta(a_t|s_t)}{\prod_{t=1}^{T}\pi(a_t|s_t)}
\end{aligned} \tag{2.49}$$

Similarly, we can derive an off-policy formulation using IS. The goal is to estimate the value of some new parameters θ':

$$J(\theta') = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau}r(\tau)]. \tag{2.50}$$

Using the IS formulation (2.48) and the identity $\pi_\theta(\tau)\nabla_\theta \log \pi\theta(\tau) = \nabla_\theta\pi_\theta(\tau)$, we can write the gradient of (2.50) as:

$$\nabla_{\theta'}J(\theta') = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\frac{\nabla_{\theta'}\pi_{\theta'}(\tau)}{\pi_\theta(\tau)}r(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)}\nabla_{\theta'}\log\pi_{\theta'}(\tau)r(\tau)] \tag{2.51}$$

If we do not want to just recover the on-policy gradient, and we actually want an off-policy algorithm, it means that we have to take multiple gradient steps without generating new samples. We will then have to use the following importance weight which is just the ratio of the products of the action probabilities:

$$\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} = \frac{\prod_{t=1}^{T} \pi_{\theta'}(a_t|s_t)}{\prod_{t=1}^{T} \pi_\theta(a_t|s_t)} \tag{2.52}$$

For policy gradient methods where we have $\theta^* = \arg\max_\theta J(\theta)$ and $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)[r(\tau)]}$, the resulting off-policy gradient is derived as:

$$\begin{aligned}
\nabla_{\theta'} J(\theta') &= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\frac{\pi'_\theta(\tau)}{\pi_\theta(\tau)}\nabla_{\theta'}\log\pi_{\theta'}(\tau)r(\tau)\right], \text{ (when } \theta \neq \theta') \\
&= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\left(\Pi_{t=1}^{T}\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}\right)\left(\sum_{t=1}^{T}\nabla_{\theta'}\log\pi_{\theta'}(a_t|s_t)\right)\left(\sum_{t=1}^{T} r(s_t,a_t)\right)\right] \text{ (by adding causality)} \\
&= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{T}\nabla_{\theta'}\log\pi_{\theta'}(a_s|s_t)\left(\Pi_{t'=1}^{t}\frac{\pi_{\theta'}(a_{t'}|s_{t'})}{\pi_\theta(a_{t'}|s_{t'})}\right)\left(\sum_{t'=t}^{T} r(s_{t'},a_{t'})\left(\Pi_{t''=t}^{t'}\frac{\pi_{\theta'}(a_{t''}|s_{t''})}{\pi_\theta(a_{t''}|s_{t''})}\right)\right)\right].
\end{aligned} \tag{2.53}$$

It is very desirable to delete the importance weight on the future rewards represented in red in Eq. (2.53). The resulting gradient is biased as equality is not hold anymore when doing that, but it turns out to still be a valid gradient. Getting rid of those additional importance weights is highly advantageous as it results in not having to multiply so many numbers together. However, we can't get rid of importance weights from timestep 1 until timestep t, which means those importance weights quickly go to 0 or infinity. In order to alleviate that problem, we can rewrite the objective a bit differently. Instead of writing it as an expectation over trajectories, we can write it as an expectation over state-action marginals (as long as they are sampled from the marginal this is a correct estimator). Fortunately, if we use this approximated gradient, we still guarantee the policy improvement and eventually achieve the true local minimum which is justified in the proof [DWS12].

The off-policy gradient is now given by:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_{i,t}|s_{i,t})\hat{Q}_{i,t}. \tag{2.54}$$

With this formulation (2.54), we can optimize the target policy by using samples from any policies.

The off-policy formulation of the policy gradient (2.54) is the central component of the RL methods used in this thesis. It allows us to improve a target policy by using samples generated by any other policies. This procedure has two main benefits. Firstly, the off-policy gradient allows an improved exploration ability as any policy can be used to explore the environment and the associated interactions can still be used for the TD learning procedure. In other words, we are not constrained to explore again the environment as soon as an update is performed. This results in a much more sample-efficient algorithm (compared to on-policy methods) as we can now hold in memory these diverse interactions and perform the updates at any desired rate. The state-of-the-art method to achieve this type of off-policy TD learning is called Experience Replay and it will be presented in Section 2.2.6. Secondly, as we can use samples generated by any policies, a nonlinear function approximator can be effectively used to estimate the values and policy functions. In fact, this kind of function approximator requires the use of uncorrelated samples so as to be optimized properly and not get stuck in local optima. In Section 2.2.7 we present how ANNs can be used as function approximators in the Actor-Critic algorithm and how they can be optimized using the Gradient Descent method.

2.2.5 The exploration-exploitation tradeoff

The deep policy gradient methods presented earlier are all based on the standard estimate of the policy gradient (2.45). In this case, the distribution representing the action distribution is updated such as increasing the probability of actions that hold a Q-Value higher than the average one and reducing it otherwise. In other words, policy gradient methods do not learn what *good actions are*, but rather which *actions are better* than the others. Therefore, the improvement ability of these algorithms directly depends on the capacity to take actions that are different from the ones suggested by the policy. This process is known as exploration while following the current policy is known as exploitation. This problem is known as the exploration-exploitation tradeoff and is a critical topic in RL: an agent needs relevant experiences to learn a good policy, but it also needs a good policy to obtain those experiences. We would like our agent to find the best actions as fast as possible, but committing to solutions too quickly and without enough exploration can be dangerous as it can lead to local minima or total failure.

Depending on the nature of the MDP, there exist optimal or suboptimal solutions to the exploration-exploitation problem which guarantee the convergence to the optimal actions. We list below methods that can be used successfully where the action and state spaces are discrete:

- **Epsilon-greedy**: the agent executes random actions randomly with some probability $\epsilon < 1$ while following the policy most of the time with probability $1 - \epsilon$. This is the most used exploration strategy thanks to its simplicity. However, with epsilon-greedy, we can keep exploring known actions, which can be in the worst case bad actions. To avoid such inefficient exploration, one idea is to reduce the parameter ϵ in time toward 0 when the optimal actions have been explored. Therefore, the final policy is almost always suboptimal, and it is difficult to determine beforehand when the parameter ϵ should be annealed. The following methods aims at automatically adjust the amount of exploration by keeping track of the occurrence of actions and their associated value.
- **Upper Confidence Bounds (UCB)**: this method relies on a basic idea that is to favor the exploration of actions with a strong potential to have an optimal value. With UPB, this potential is measured by the upper bound confidence bound of the reward value $\hat{U}_t(a)$, so that the true value is below with bound $Q_t(a) \leq \hat{Q}_t(a) - \hat{U}_t(a)$ with high probability. The upper bound $\hat{U}_t(a)$ is a function of $N_t(a)$ a large number of trials which ensure a smaller bound $\hat{U}_t(a)$. This is possible using Hoeffding's Inequality allowing us to not assign any prior knowledge on how the distributions look like. The agent selects the greediest action to maximize the upper confidence bound: $a_t^{UCB} = \arg\max_{a \in A} \hat{Q}_t(a) + \hat{U}_t(a)$.
- **Bayesian UBC**: in UCB we don't consider any prior on the shape of the reward distribution and therefore we rely on Hoeffding's Inequality to have a very general estimation by considering every possibility. If we could have the shape of this distribution beforehand, we could have a better estimation. If for example, we expect the distribution to be a Gaussian, we can set the upper bound confidence as 95% confidence interval by setting $U_t(a)$ to be twice the standard deviation.
- **Boltzmann exploration**: we would like to use all the information encompass in our Q-Value function estimate. Instead of taking the optimal action or a random one, Boltzmann exploration [Ces+17] involves choosing an action with weighted probabilities. Thus, the softmax function is used over the Q-Value function estimate. Therefore, the action which the estimate believe is optimal is more likely, but not guaranteed, to be chosen. The advantage of Boltzmann's exploration is that information about the value of other actions is taken into consideration in the relative weight. This way the agent ignores actions that are estimated to be largely suboptimal while giving more attention to potentially promising, but not necessarily ideal actions. In practice, a parameter τ is introduced. It is annealed over time and it controls how spread is the softmax distribution so that all actions are considered equally at the beginning and sparsely distributed by the end of training. The action sampling process is then defined as:
$$\pi(a|h_t) = \frac{\exp\{[Q_t(a)/\tau]\}}{\sum_{i=1}^{n} \exp\{[Q_t(i)/\tau]\}}. \tag{2.55}$$
- **Thompson sampling**: the agent keeps track of a belief over the probability of optimal actions and samples from this distribution. The action sampling process is here defined as:
$$\begin{aligned} \pi(a|h_t) &= \mathbb{P}[Q_t(a) > Q(a'), \forall a' \neq a | h_t] \\ &= \mathbb{E}_{\mathcal{R}|h_t}[\mathbb{1}(a = \arg\max_{a \in A} Q_t(a))], \end{aligned} \tag{2.56}$$
where $\pi(a; |h_t)$ is the probability of taking action a given the history h_t. The Thompson sampling strategy is based on the *Probability Matching* decision-making strategy in the stochastic context where the probability of actions matches the probability of reward. It can be computationally intractable to estimate the posterior distributions, but Thompson sampling can still work using methods like Gibbs sampling, Laplace approximate, and the bootstraps for the approximation process.

The convergence to the optimal policy of the previous methods is guaranteed only when the action and state spaces are discrete. When dealing with continuous spaces, the benefits of these methods do not hold anymore as the probability to visit the exact same state twice is in this case null. Other methods have to be used for improved exploration when using ANNs. The problem of exploration is a complete field itself (as illustrated by the "Noisy-TV" problem [Bur+19]) and extensive work is currently done toward the development of efficient methods. In the following, we will only focus on exploration methods for deep policy gradient methods. When the action and state spaces are continuous, the exploration-exploitation problem becomes untrackable and the true optimal solutions can not be obtained. In our case of AUV applications, the number of training trials, despite being mostly simulated, is closer to being finite than infinite. Therefore, given a relatively small amount of interactions, we need to explore effectively these spaces in order to build an appropriate policy. The exploration strategies can then be decomposed into two groups: direct and indirect exploration. With direct exploration, the policy outputs are modified before being applied. The most simple way of doing that remains the epsilon-greedy strategy described earlier. However, epsilon-greedy is not effective to explore continuous space (where there is an infinite set of possibilities), an improved practice consists instead in applying noise to the actions estimated by the policy network. The action sampling is then defined as:

$$a_t = \pi(a_t|s_t) + \mathcal{N}(0, \sigma), \tag{2.57}$$

where $\mathcal{N}(0, \sigma)$ is a normal noise of mean 0 and standard deviation σ. Depending on the dynamics of the process, the noise standard deviation requires careful tuning as a too-small value can result in poor exploration and failure at performance improvement, while a too high value can result in high variance and restrain the agent from completing the task. Similar to Boltzmann's exploration [Ces+17], the noise standard deviation could also be adapted over the course of training. In indirect exploration, the idea is to introduce noise before the action sampling process. This type of exploration is more appropriate for robotic applications as the resulting actions will be correlated to the agent's perception of the environment, and therefore they will be related to the dynamics of the process rather than being purely random and uncorrelated. The actions resulting from epsilon-greedy are totally random and uncorrelated to the agent's state, while with noise induced for example in the state, observation, or even the parameter space, the resulting actions will be correlated to the environment dynamics. One approach consists in changing the overall objective function, exactly as performed within the maximum entropy reinforcement learning framework presented in Section 2.2.8. By adding the entropy term in the policy loss function, the agent is forced to explore more equally and take more diverse actions. Inspired by Thompson sampling, Bootstrapped DQN [Osb+16] introduces a notion of uncertainty in Q-value approximation in the DQN algorithm by using the bootstrapping method. Bootstrapping is to approximate a distribution by sampling with replacement from the same population multiple times and then aggregating the results. The idea is to have multiple Q-value estimators trained in parallel but each only consumes a bootstrapped sub-sampled set of data and each has its own corresponding target network. All the Q-value heads share the same backbone network. At the beginning of one episode, one Q-value estimator is sampled uniformly and acts for collecting experience data in this episode. Then a binary mask is sampled from the masking distribution $m \sim M$ and decides which estimator can use this data for training.

Nevertheless, this kind of indirect exploration is still restricted because the induced noise and uncertainty rely on the training data. We can inject some prior information independent of the data by using parameter noise [Pla+18]. Parameter noise adds adaptive noise to the parameters of the policy neural network policy, rather than to its action space. It injects randomness directly into the parameters of the agent, altering the types of decisions it makes such that they always fully depend on what the agent currently senses. The technique is a middle ground between evolution strategies (where you manipulate the parameters of your policy but don't influence the actions a policy takes as it explores the environment during each rollout) and deep reinforcement learning approaches like TRPO, DQN, and DDPG (where we add noise on the action). In practice, random Gaussian noise $\mathcal{N}(0, \sigma)$ is added to the parameters of the policy network at the beginning of each episode (then kept during the rollout) as:

$$\sigma_{k+1} = \begin{cases} \alpha\sigma_k, & \textbf{if } d(\pi, \tilde{\pi}) < \delta, \\ \frac{1}{\alpha}\sigma_k, & \textbf{otherwise}. \end{cases} \tag{2.58}$$

The noise standard deviation σ is adapted according to a distance measure $d(\cdot)$ between the non-perturbed π and perturbed policy $\tilde{\pi}$ which is defined in [Pla+18] as:

$$d(\pi, \tilde{\pi}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_s[(\tilde{\pi}(s)_i - \pi(s)_i)^2]}, \tag{2.59}$$

where the metric $\mathbb{E}_s[\cdot]$ is estimated over a large number of samples from the Replay Buffer and N is the dimension of the action space. We can directly see that by setting $\delta = \sigma$ (2.58), it is equivalent to an action space noise $\mathcal{N}(0, \sigma)$. Now that we have seen how to interact with the environment to address the exploration-exploitation tradeoff, we will present in the next section how to use these generated data to improve our value estimates and policy.

2.2.6 Experience replay

The concept of Experience Replay (ER) [Lin04] employs the agent's past experience to improve its current behavior. It aims to artificially make the agent's experience look Independent and Identically Distributed (IID). This is highly desirable in order to not concentrate the updates to a limited area of the desired functions. Given that an agent's experience at the timestep t is defined as the quadruplet $e_t = (s_t, a_t, r_t, s_{t+1})$, the ER method consists in storing (at each timestep) the experience e_t in a memory unit $\mathcal{D} = \{e_1, \ldots, e_t\}$ of fixed size, also known as the *replay buffer*. Then, the neural networks are trained by performing mini-batch gradient descent of past experiences randomly sampled from the replay buffer. The estimators are hence trained on IID samples that are generated by various trajectories and policies and they are, therefore, more representative of the true function.

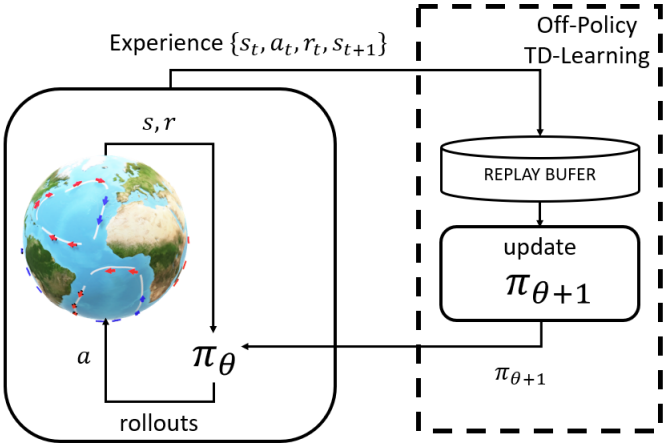


Figure 2.15: *Illustration of an off-policy TD-Learning procedure where the policy used for exploration is different from the one that is optimized.*

As illustrated in Figure 2.2.6, in off-policy policy gradient methods such as the Actor-Critic algorithm, the policy used for data collection is different from the one that generated the samples used for off-policy TD-Learning. The usage of ER can be summarized as follow:

1. Initialize s, w and θ
2. For $t = 1, \ldots, T$:
   (a) Sample reward $r_t \sim R(s, a)$ and the next state $s' \sim P(s'|s, a)$.
   (b) Sample the next action performed $a' \sim \pi(s', a'; \theta)$.
   (c) Update the policy parameters as: $\theta \leftarrow \theta + \alpha_\theta Q(s, a; w) \nabla_\theta \log \pi(a|s; \theta)$.
   (d) Compute the TD error at timestep t as: $G_{t:t+1} = r_t + \gamma Q(s', a'; w) - Q(s, a; w)$.
   (e) Update the parameters w according to this TD error: $w \leftarrow w + \alpha_w G_{t:t+1} \nabla_w Q(s, a; w)$.
   (f) Move to the next transition $a \leftarrow a'$ and $s \leftarrow s'$.

There are multiple benefits to using experience replay. By sampling at random, we increase the probability that our updates to the neural network will have less variance. In On-policy TD learning, most of the samples used for gradient descent were correlated and similar. Updating with similar samples concentrates the updates we make to our neural network to a limited area of our function, and it potentially over-emphasizes the magnitude of the updates. If we sample uniformly at random from a very large buffer, on the other hand, chances are our updates to the network will be better distributed all across, and therefore more representative of the true function. Using a replay buffer also gives the impression our data is IID, so optimization methods will be better behaved. Samples will seem independent and identically distributed because we will be sampling from multiple trajectories and policies at once.

2.2.7 Deep Policy Gradient

In robotic applications, we use sensors to measure the environment, and we want to use the measured data to perform informed decisions based on what the sensors observe. Therefore, the goal is to build a policy function that takes as input the sensor data and returns the best actions. There is a large number of methods to find patterns in sensor data. For example, we might just use linear regression to determine how data is trending over time. Or we may look for when the signal drops below some threshold to indicate that whatever the sensor was observing is no longer in view. Nevertheless, the complexity of these methods, and how difficult it is to develop, scales with the complexity of the pattern that we are trying to find, as well as how those patterns vary from one observation to another. Using rule-based approaches, the defining features are difficult to describe.

In addition, the policy, value functions, and model can all be represented by a parametric function. For discrete spaces, in theory, we can memorize $Q_*(\cdot)$ for all state-action pairs, like an enormous lookup table. However, it quickly becomes computationally infeasible when the state and action space are large, or simply impossible when they are continuous. Moreover, when using ANNs as function approximators for the policy, we can consider stochastic actors that are better at rejecting disturbance compare to their deterministic counterparts. This is possible thanks to the reparameterization trick that will be presented in Section 2.2.8.

For these reasons, a common practice consists in using function approximators (i.e. machine learning models) to learn them from experience. Among the various methods of function approximation, artificial neural networks (ANNs) are leading the field. When multiple layers are used, people typically refer to this as deep reinforcement learning. In this section, we will thus focus on deep policy gradient methods which use ANNs to approximate the value and policy functions.

Artificial neural networks

Deep reinforcement learning can be applied to solve very large problems such as:

- Backgammon [Tes94]: 10^{20} states
- Go [Sil+16]: 10^{170} states
- Helicopter [Ng+03; Ng+04]: continuous state space
- Robots [Gu+17]: limited view of the world

So far, we have only considered lookup tables to represent the value function where every state s has an entry $v(s)$ or every state-action pair (s, a) has an entry $Q(s, a)$. The resulting problem with large MDP, which we want to tackle with ANNs, is obviously that there are so many states in the lookup tables that can not fit the memory in the computer. We can not build a table that covers a game like Go for instance. Additionally, and sometimes even more importantly, it is too slow to try to learn for each state individually. This is essentially the problem of generalization, as whenever we observe a state, we want to learn about that state, but we also want to learn about all the similar states because if it is a very big problem or continuous space, we will never see the exact same state twice. So in this case, if we would learn about each state individually, we would never learn anything, or we would never learn anything that we can use in the latter states because they would all look like brand new states. Another challenge is that individual states are often not fully observable. The perception of the world by robots is limited by their sensors and we want to somewhat deal with that.

There are many function approximators such as ANN, Decision Tree, Nearest neighbor, Fourier / Wavelet bases or Coarse coding, but reinforcement learning has specific properties:

- The agent's experience is not Independent and Identically Distributed (IID). Typically, successive timesteps are correlated.
- The agent's policy affects the data it receives which affects the nature of the function we are learning.
- The value function $V(s)$ can be non-stationary. Policy Gradient methods are a very clear example of that as we change the policy repeatedly because we want to build a policy that works better, which means each time we are trying to estimate a different value function. Depending on the algorithm it can still be stationary, for instance when using bootstrapping with TD Learning.
- Feedback is delayed. In the online case, we might execute an action, and immediately update using TD Learning, but it is not clear that it is always the best thing to do. Sometimes we want to wait for a few updates as in Monte-Carlo methods, which is not always trivial to do.

For these reasons, differentiable (nonlinear) function approximation (ANNs) is preferred in the field of RL and often performs best. Let's now describe how such a function approximator can be optimized using the Gradient Descent approach.

Let's consider $J(\theta)$ to be a differentiable function of parameter vector θ. The gradient of $J(\theta)$ is given by:

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \cdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix} \tag{2.60}$$

The goal will be to find a minimum of $J(\theta)$ (if it is considered as a loss) by moving the parameters in the direction of the negative gradients to perform Gradient Descent:

$$\boldsymbol{\Delta}\theta = -\frac{1}{2}\alpha \nabla_\theta J(\theta), \tag{2.61}$$

where α is a step-size parameter.

For illustration, now let's see how we can approximate values by using stochastic gradient descent in order to use ANNs as function approximators. The goal is then to find θ that minimizes the difference between $V_\theta(s)$ and $V_\pi(s)$:

$$J(\theta) = \mathbb{E}_\pi \left[(V_\pi(S) - V_\theta(S))^2 \right], \tag{2.62}$$

with Gradient Descent the update is derived as:

$$\begin{aligned} \boldsymbol{\Delta}\theta &= -\frac{1}{2}\alpha \nabla_\theta J(\theta), \\ &= \alpha \mathbb{E}_\pi \left[(V_\pi(S)) - V_\theta(S) \nabla_\theta V_\theta(S) \right]. \end{aligned} \tag{2.63}$$

In this particular formulation (2.63), the only thing that is random is the state, because we are using here the true value function which of course is not something we can do in practice but we do it to just define the loss. With Stochastic Gradient Descent, the update is given by:

$$\boldsymbol{\Delta}\theta_t = \alpha (V_\pi(S_t) - V_\theta(S_t)) \nabla_\theta V_\theta(S_t). \tag{2.64}$$

In this case (2.64), this sampled gradient is still not something that is available because although we have instantiated the states, the true function is still not instantiated, which we do not have. Noted that here we assumed the policy is fixed and does not depend on the value function, therefore these updates are only considering policy evaluation and do not cover policy improvement.

To make this approach more concrete, we present now the loss functions used to design a deep Actor-Critic method. The Q-Value function does not require an estimation of the State-Value function, but as some standard methods still estimate them together for stability purposes, we will define their associated loss function. First, the State-Value function is estimated by an ANN parameterized by Ψ. In order to reduce Actor-Critic value overestimation [HGS16; FHM18], the State-Value function is estimated using the minimum of two different Q-Value estimates represented by two ANNs parameterized by Υ_1 and Υ_2, respectively. We use TD learning to estimate these functions, thus Ψ are optimized to minimize the TD error:

$$J_V(\Psi) = V_\Psi^{\pi_\mu}(s_t) - \left(\min \left[Q_{\Upsilon_1}^{\pi_\mu}(s_t, a_t), Q_{\Upsilon_2}^{\pi_\mu}(s_t, a_t) \right] \right). \tag{2.65}$$

Similarly, the parameters Υ_i of the i-th Q-Value function estimator are optimized to minimize the TD error:

$$J_Q(\Upsilon_i) = Q_{\Upsilon_i}^{\pi_\mu}(s_t, a_t) - \left(r(s_t, a_t) + \gamma \times V_{\Psi'}^{\pi_\mu}(s_{t+1}) \right), \tag{2.66}$$

where Ψ' are the parameters of the bootstrapped state-value function, denoted as the target value (defined later in Section 2.2.8) and γ is the discount factor 2.2.1. Finally, the parameters θ of the policy π_θ are updated as follows:

$$\theta \leftarrow \theta + \alpha \min[Q_{\Upsilon_1}^{\pi_\mu}(s_t, a_t), Q_{\Upsilon_2}^{\pi_\mu}(s_t, a_t)] \nabla_\theta \log \pi_\theta(a_t | s_t), \tag{2.67}$$

where α is a learning rate. When considering these loss functions (2.65)(2.66)(2.67), the ANNS are optimized exactly for the purpose of estimating the Bellman's equations (2.26)(2.27) and the vanilla policy gradient (2.43) respectively. Using experience replay, the samples are no longer correlated and we have multiple samples in the batch, so we no longer need to perform these single-sample SGD updates.

2.2.8 Maximum entropy RL

When an agent is exploring the environment, it picks the next actions to execute based on its current belief (i.e. Q-Value estimate). Without enough exploration, an agent can easily get stuck in local minima and never experience satisfying trajectories. This behavior is illustrated in Figure 2.16 below. In this example, the goal of the agent is, from a starting position represented in green, to reach the target position (represented by the life lifebuoy) that is accessible only by rounding the container ship to the right. Now let's imagine that for some reason due to randomness, the agent has initially passed more often and deeper into the left path. At this time, the Q-value estimate in the left region will then be the highest, because, despite the goal not being reachable from there, we can more easily get closer to it (especially if the reward is for example a function of the Euclidean distance to the target). Thus, the policy gradient (2.45) will encourage the agent to further explore the left path, such as the right path is not explored anymore, which can happen quickly.

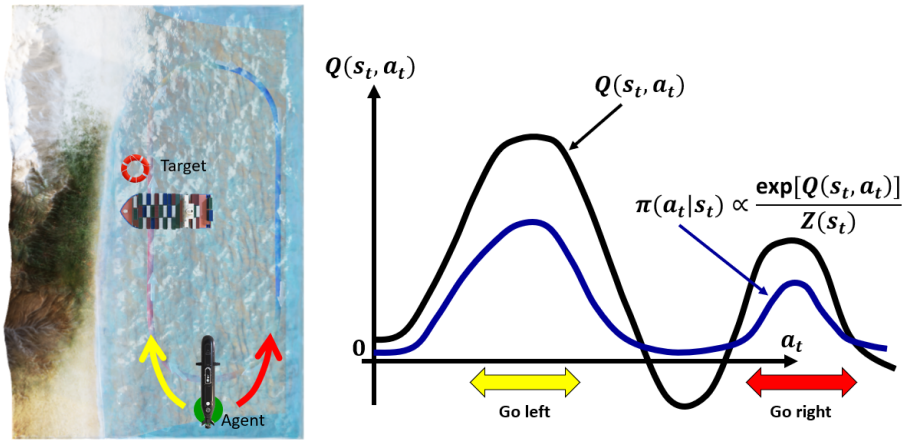


Figure 2.16: *In maximum entropy reinforcement learning the policy is forced to explore the space of Q-Value more equally. This results in improved robustness to process variation as the agent is forced to explore sub-optimal strategies until the optimal long-term one is found.*

We would like a framework allowing us to systematically explore both paths such that the target is reached at least once. In order to improve the exploration balance between the two paths, we would be interested in transforming the Q-Value distribution so as lower values (and negative ones in particular) will still induce some exploration by the policy gradient. A simple transformation performing exactly this is the exponential transformation (illustrated by the blue curve in Figure 2.16). As the policy is represented by an ANN, we can optimize our policy exactly for this goal by minimizing the Kullback–Leibler (KL) divergence [Joy11] between the policy distribution and the exponential of the Q-value distribution:

$$\min_{\pi} DKL(\pi(\cdot|s_0)||\exp\{Q(s_0,\cdot)\})) \tag{2.68}$$

Nonetheless, if we write the definition of the KL divergence, up to an additive constant, this objective function (2.68) is then equal to the expectation under π of $Q(\cdot)$ minus log of π:

$$J_{MaxEnt}(\pi|s_0) = \max_{\pi} \mathbb{E}_{\pi}[Q(s_0,a_0) - \log \pi(a_0|s_0)]. \tag{2.69}$$

When we look at the objective function (2.69), we can simply decompose the first term as the reward and the second term is just the entropy of the policy:

$$J_{MaxEnt}(\pi|s_0) = \max_{\pi} \mathbb{E}_{\pi}[\sum_{t} r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\big|s_0], \tag{2.70}$$

where the term in green is known as the Shannon entropy measure, weighted by a temperature parameter α. This formulation (2.70) looks similar to the original Reinforcement Learning objective (2.17) [SB18] except that it has now an entropy term in there. This function is called the maximum entropy objective $J_{MaxEnt}(\pi|s_0)$. Solution methods using the objective function (2.70) with that addition term are denoted as maximum entropy reinforcement learning method. There exist different ways to measure a distribution entropy, but Shannon's entropy is commonly used as it is easy to compute. The term "entropy" in thermodynamics and information theory both captures increasing randomness. The maximum entropy objective function is quite interesting as we are asking the agent to be as random as possible while also maximizing the reward. If we optimize this objective function (2.70), the agent will look for all the different ways to maximize the reward. It will undoubtedly explore both left and right passages until it figures out which one is really better and leads to the target waypoint. In the following, we present an Actor-Critic algorithm that takes into account this entropy term.

Soft Actor-Critic

Let x be a random variable with probability mass or density function P. The entropy $\mathcal{H}$ of x is computed from its distribution P according to:

$$\mathcal{H}(x) = \mathbb{E}_{x\sim P}[-\log(x)]. \tag{2.71}$$

In practice, following this expression (2.71) the entropy term is explicitly incorporated in the State-Value function $V(s_t)$ as:

$$\begin{aligned} V(s_t) &= \mathbb{E}[Q(s_t,a_t) + \alpha\mathcal{H}(\pi_\mu(.|s_t))], \\ &= \mathbb{E}[Q(s_t,a_t) - \alpha\log\pi_\mu(a_t|s_t)], \end{aligned} \tag{2.72}$$

where α is controlled either directly or indirectly and affects the stochasticity of the resulting policy because it serves the role of the temperature of the energy-based optimal policy [Haa+17]. With this modification, the expected return of the agent does not encompass only the reward but also the entropy of the policy. The parameters Ψ of the State-Value function are then optimized to minimize the TD error:

$$J_V(\Psi) = V_\Psi^{\pi_\mu}(s_t) - (\min\left[Q_{\Upsilon_1}^{\pi_\mu}(s_t,a_t), Q_{\Upsilon_2}^{\pi_\mu}(s_t,a_t)\right] - \log\pi_\mu(.|s_t)). \tag{2.73}$$

Similarly, the parameters Υ_i of the i-th Q-Value function estimator are optimized to minimize the TD error:

$$J_Q(\Upsilon_i) = Q_{\Upsilon_i}^{\pi_\mu}(s_t,a_t) - \left(r(s_t,a_t) + \gamma \times V_{\Psi'}^{\pi_\mu}(s_{t+1})\right), \tag{2.74}$$

Finally, the parameters μ of the Policy network are then optimized in order to minimize the expected Kullback-Leibler (KL) divergence between the current policy and the exponential of the Q-Value function that is normalized by a function Z_Υ according to [Haa+18c] as:

$$J_\pi(\mu) = \mathbb{E}_{s_t\sim D}\left[D_{KL}\left(\pi_\mu(\cdot|s_t)\middle|\middle|\frac{Q^*(s_t,\cdot)}{Z_\Upsilon(s_t)}\right)\right], \tag{2.75}$$

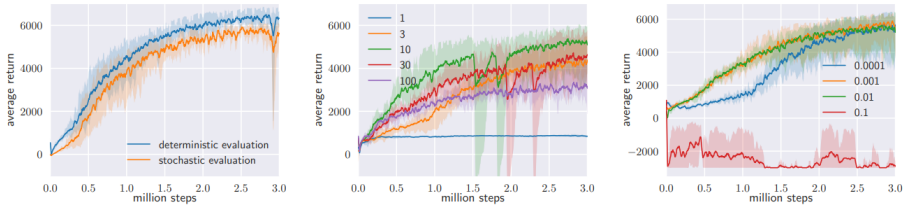
where,

$$Q^*(s_t,a_t) = \exp\left(\min\left[Q_{\Upsilon_1}^{\pi_\mu}(s_t,a_t), Q_{\Upsilon_2}^{\pi_\mu}(s_t,a_t)\right]\right), \tag{2.76}$$

where $Z_\Upsilon(s_t)$ is a partition function used to normalize the resulting distribution. While $Z(\cdot)$ is generally intractable, it does not contribute to the gradient with respect to the new policy and therefore can be ignored. When using the distribution expressed in Eq. (2.76) as a target for the policy shown in Eq. (2.75), the agent is forced to explore actions proportionally to their associated exponential Q-Values. This positive transformation allows a smarter exploration-exploitation tradeoff as negative Q-Values will be transformed into small but positive ones, forcing the policy to make progress along sub-optimal strategies. The overall algorithm is called **Soft Actor-Critic (SAC)** [Haa+18c] which provides an unbiased estimator of the gradient of (2.75) as:

$$\hat{\nabla}_\mu J_\pi(\mu) = \nabla_\mu\log\pi_\mu(a_t|s_t) + \left(\nabla_{a_t}\log\pi_\mu(a_t|s_t) - \nabla_{a_t}\min(Q_{\Upsilon_1}^{\pi_\mu}(s_t,a_t), Q_{\Upsilon_2}^{\pi_\mu}(s_t,a_t))\nabla_\mu f_\mu(\epsilon_t,s_t)\right). \tag{2.77}$$

The derivative expressed in Eq. (2.77) allows the use of Gradient Descent to optimize the parameters μ of the Policy ANN. Considering Eq. (2.72), the parameters μ are consequently optimized exactly for the desired maximum entropy objective (2.70). The soft Q-update (2.74) guarantees that $Q^{\pi_{new}}(s_t,a_t) \geq Q^{\pi_{old}}(s_t,a_t)$ and the repeated policy updates (2.77) ensure convergence toward the optimal policy π^* (see Appendix B.2 and B.3 of [Haa+18c] for the mathematical proof). The parameter α in (2.72) controls the relative weight of the entropy term against the Q-Value in the State-Value function. As illustrated in Figure 2.17, for small α magnitudes, the policy becomes nearly uniform, and consequently fails to exploit the reward signal, resulting in substantial degradation of performance. For large α magnitudes, the policy learns quickly at first, but the policy then becomes nearly deterministic, leading to poor local minima due to a lack of adequate exploration. With the right reward scaling, the agent balances exploration and exploitation, leading to faster learning and better asymptotic performance. Next, we will present an improvement of the SAC algorithm which consists in adjusting automatically the temperature parameter α.



(a) Evaluation (b) Reward Scale (c) Target Smoothing Coefficient (τ)

Figure 2.17: *Illustration from [Haa+18c] of the sensitivity of the nominal SAC algorithm on the AntV1 task from the OpenAI gym benchmark suite [Bro+16].*

Automatic entropy adjustment

The optimal reward scale is difficult to determine beforehand because the entropy term in the objective function (2.70) can greatly vary across tasks and during training as the policy becomes better. An improvement of the SAC algorithms consists of adapting the temperature term so as to maintain the desired entropy value:

$$\max_{\pi_0,\dots,\pi_T} \mathbb{E}[\sum_{t=0}^{T} r(s_t, a_t)] \text{ s.t. } \forall t,\ \mathcal{H}(\pi_t) \geq \mathcal{H}_0. \tag{2.78}$$

The expected return $\mathbb{E}[\sum_{t=0}^{T} r(s_t, a_t)]$ can be decomposed into a sum of rewards at all the timesteps. Because the policy π_t at time t has no effect on the policy at the earlier timestep, π_{t-1}, we can maximize the return at different steps backward in time:

$$\underbrace{\max_{\pi_0}\Big(\mathbb{E}[r(s_0,a_0)] + \underbrace{\max_{\pi_1}\big(\mathbb{E}[...] + \underbrace{\max_{\pi_T}\mathbb{E}[r(s_T,a_T)]}_{\text{1st maximization}}\big)}_{\text{second but last maximization}}\Big)}_{\text{last maximization}} \tag{2.79}$$

where we consider $\gamma = 1$ (this is essentially a DP process as described in Section 2.2.4). The practical algorithm was proposed in [Haa+18a] and is presented now: The Q-Value function is estimated exactly as proposed in the first version of SAC by minimizing the TD error (2.74). On the other hand, the State-Value function is now not explicitly represented by a neural network, but it is implicitly defined through the Q-Value functions and the policy (as no differences are observed when comparing both methodologies):

$$V(s) \simeq \mathbb{E}_{(s_t\sim D, a_t\sim\pi_\mu)}\big[\min_{i\in 1,2} Q^{\pi_\mu}_{\Upsilon_i}(s_t,a_t) - \alpha\log\pi_\mu(a_t|s_t)\big]. \tag{2.80}$$

Using the reparameterization trick provided in Section 2.2.8, the Gaussian policy is learned by minimizing the following cost function:

$$J_\pi(\mu) = \mathbb{E}_{s_t\ D, a_t\ \pi_\mu}[\alpha\log\pi_\mu(a_t|s_t) - \min_{i\in 1,2} Q^{\pi_\mu}_{\Upsilon_i}(s_t,a_t)]. \tag{2.81}$$

This procedure Eqs. (2.80)(2.81) is the same as the first version of SAC but with an explicit, dynamic temperature parameter α. To learn α, we have to minimize the dual objective (2.78), which is possible by using dual gradient descent. Instead of optimizing with respect to both variables (i.e. reward and desired entropy), the authors [Haa+18a] proposed to perform incomplete optimization that alternates between taking a single gradient step on each objective successively. By doing this, the resulting loss function can now be used with mini-batch gradient descent. Therefore, the gradients for α are computed with the following objective function:

$$J(\alpha) = \mathbb{E}_{s_t\ D, a_t\ \pi_\mu}\big[-\alpha\log\pi_\mu(a_t|s_t) - \alpha\mathcal{H}\big]. \tag{2.82}$$

Finally, in order to make sure that α (2.82) is non-negative, in practice we parameterize $\alpha_t = \exp\{\beta_t\}$ and we optimize β_t instead. This results in a maximum entropy policy gradient method where the reward scale does not need to be tuned. The target entropy is more intuitive to tune and is traditionally set to $\mathcal{H} = -dim(u)$ which means that for each degree of freedom, the entropy term in the objective function will hold a relative weight of 1% against the reward term. In order to update the policy, we need to draw a sample by computing a deterministic function of the state, policy parameters, and independent noise. This is possible using the reparameterization trick that we present next.

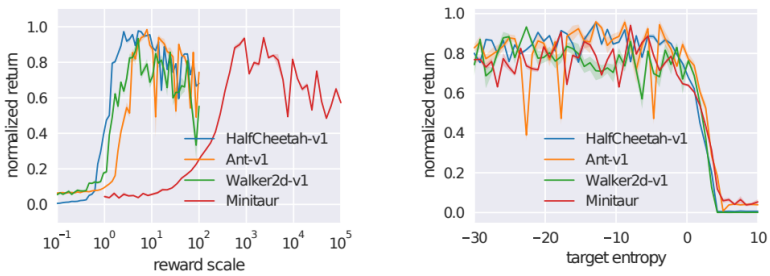


Figure 2.18: *Illustration from [Haa+18a] of the performance variation observed depending on the version of the SAC. The performance is more sensitive to the reward scale with a fixed temperature parameter (left) compare to when it is adjusted automatically (right).*

Reparameterization trick

In order to optimize the policy for both versions of the SAC algorithm, we need to use the reparameterization trick. It allows us to express the expectation over actions into an expectation over the noise. By doing so, the probability distribution representing the action distribution has no dependence on the policy parameters. The reparameterization is defined as:

$$\mathbb{E}_{a\sim\pi_\mu}\left[Q^{\pi_\mu}(s,a) - \alpha\log\pi_\mu(a|s)\right] = \mathbb{E}_{\xi\sim\mathcal{N}}\left[Q^{\pi_\mu}(s,\tilde{a}_\mu(s,\xi)) - \alpha\log\pi_\mu(\tilde{a}_\mu(s,\xi)|s)\right]. \tag{2.83}$$

To derive the policy loss, the final step is to substitute the Q-Value function with one of the function approximators. In the case of SAC, unlike TD3 and DDPG, the minimum of two Q-Value approximators is exploited. The policy is thus optimized according to:

$$\max_\mu \mathbb{E}_{s\sim D,\xi\sim\mathcal{N}}\left[\min_{j=1,2} Q_{\Upsilon_i}(s,\tilde{a}_\mu(s,\xi)) - \alpha\log\pi_\mu(\tilde{a}_\mu(s,\xi)|s)\right], \tag{2.84}$$

which is almost exactly the same as the DDPG and TD3 policy optimization procedure, except for the minimum of the double Q-Value approximators, the stochasticity of the policy, and the entropy term.

By considering the maximum entropy reinforcement learning framework, we can build a more robust control system for the AUV. The control system will be forced during training to complete the considered task in various ways. This results in more adaptability in operation when facing a new set of states and thus not all solutions are available anymore. The exponential transformation of the space of Q-Value ensures an improved exploration of the continuous action and state spaces and reduces the chances of getting stuck in local optima. In this thesis, we use this maximum entropy reinforcement learning framework as it allows us to build an AUV control system that is more robust to process variation. Indeed, the agent is forced during training to solve the task in various ways, rather than not committing to a single appropriate solution, allowing it to be more flexible in evaluation when facing unseen scenarios. This strategy is preferred to the original RL objective that solely takes into account the reward because since we can not perform an exhaustive exploration of the action and state spaces, it is better to learn a policy that intrinsically holds some adaptation abilities (i.e. Soft Actor-Critic) rather than trying to only maximize the expected return (e.g. DDPG [Lil+16], TD3 [FHM18], TPRO [Sch+15], PPO [Sch+17], …). In addition, Deep Policy Gradient methods are generally very sensitive to the choice of hyperparameters (e.g. learning rate, mini-batch size, target update procedure, …). The Soft Actor-Critic on the other hand has shown to work properly on many environments and with the exact same hyperparameters and ANNs architecture [Haa+18c][Haa+18b][Haa+18a] which is why we choose it as the core RL algorithms for our proposed methods.

The TD learning procedure seems similar to supervised learning as the considered loss functions Eqs. (2.73) and (2.74) are composed of variables that are either stored in the Replay Buffer or that can be sampled from the estimators directly (i.e. the target values are known). However, the major difference is that these losses are functions of each other which remove all of the convergence guarantees that are well established in the context of supervised learning. This is known as the moving target problem and we present next how it is tackled in the context of Deep Reinforcement Learning.

Target values

In supervised learning, target values are the labels on the dataset that by definition are static. The weights of the ANN are then updated for the purpose of modeling an optimal nonlinear mapping function between the input vector and the corresponding target values vector. As defined earlier, in Actor-Critic methods, the value estimators are optimized in order to minimize the TD errors presented in Section 2.2.8 cf Eqs. (2.73) and (2.74). The value estimates are then used to update the Policy network so that optimal actions are taken (according to the associated optimal values Q_* and V_* that are the best return we can obtain). The target value in Eq. (2.74) is composed of the reward $r(s_t, a_t)$ generated at the evaluated state s_t for a given policy, that is stored in the Replay Buffer and thus is fixed, as well as of the expected future cumulative reward of the agent represented by $V(s_{t+1})$ which is bootstrapped using our current estimate of the State-Value function. Therefore, contrary to the supervised learning case, a part of the target values are not static. At each gradient update, this target value will change given that the weights of the critic network have been updated. In fact, each estimator Eqs. (2.73), (2.74), and (2.75) are functions of the other (directly or indirectly). At each update step, they are improved, changing the shape of the other functions. The value estimates change at each iteration, making the previous estimates invalid after each gradient update:

$$\begin{aligned} V_t^{\pi}(s_{t+1}) &\neq V_{t+1}^{\pi}(s_{t+1}) \\ Q_t^{\pi}(s_t, a_t) &\neq Q_{t+1}^{\pi}(s_t, a_t). \end{aligned} \tag{2.85}$$

The TD-Learning procedure (2.73)(2.74) is different from Gradient Descent in the sense that the target value periodically changes, transforming the loss landscape. This generates training behaviors that are unstable and divergent. To tackle this problem, it is common practice to have a separate copy of the considered value network, denoted as the "target" network, whose parameters are moving slowly or are fixed over a predefined amount of iteration steps. Here, we define a Target State-Value network $V_{\Psi'}(s)$ that is then used to compute the Q-Value TD error (2.74) as:

$$J_Q(\Upsilon_i) = Q_{\Upsilon_i}^{\pi_\mu}(s_t, a_t) - \left(r(s_t, a_t) + \gamma \times V_{\bar{\Psi}'}^{\pi_\mu}(s_{t+1})\right), \tag{2.86}$$

where the parameters Ψ' slowly track the value of Ψ. As illustrated in Figure 2.19, by having a target network we reduce the "chasing your own tail" [Mni+15] problem by artificially creating small supervised learning problems. This is illustrated in Figure 2.19, where, by fixing the target value (i.e. label), the model is able to smoothly converges toward it before it is updated. This procedure improves the chances for convergence, not to the optimal values (as this can not be the case with respect to nonlinear function approximation), but convergence in general, while reducing substantially the chances of divergence. Two strategies for target network update are defined, both leading to stable learning dynamics: hard update and soft update.

1. *Hard update*: it consists of freezing completely the target network weights and copying directly the weights of the chosen value estimators in the target network every Γ gradient step. The advantage of this procedure is that by choosing a proper Γ, we can ensure a minimal amount of residual TD error, making the overall learning process more stable as we ensure the convergence guarantees of gradient-based supervised learning. The drawback is that it significantly increases the training time as the policy is not trained on up-to-date values. In addition, the value of Γ has to be tuned according to the process complexity.
2. *Soft update*: it consists of slowly copying the weights using an exponential moving average with Δ a smoothing constant. The advantage of this procedure is that the policy update is not lagging for up to Γ timesteps as the target values move (slowly) at each update. The drawback is that residual TD error can increase over the course of training, increasing learning instability. The range of proper value for Δ is notably smaller compare to Γ and nowadays the value $\Delta = 0.005$ is the standard.

The target value trick allows us to make the TD learning procedure actually converges when using ANNs. As shown in Figure 2.19, without any target network, deep TD learning diverges which results in the failure of RL algorithms. The *soft update* method with the standard value $\Delta = 0.005$ is traditionally used in AUV applications as it ensures limited residual TD error. Nevertheless, it has also been proved [FHM18] that Value estimates diverge through overestimation when the policy is poor, and the policy will become poor if the value estimate itself is inaccurate. For this reason, the policy network should be updated at a lower frequency than the value network, to first minimize error before introducing a policy update. This is known as the *Delayed Target Update* trick (proposed in [FHM18]) and consists in updating the Critic every step while only updating the Actor and target Critic network every d iterations, with $d > 1$. By sufficiently delaying the policy updates we limit the likelihood of repeating updates with respect to an unchanged Critic. The less frequent policy updates that do occur will use a Value estimate with lower variance, which results in higher-quality policy updates.

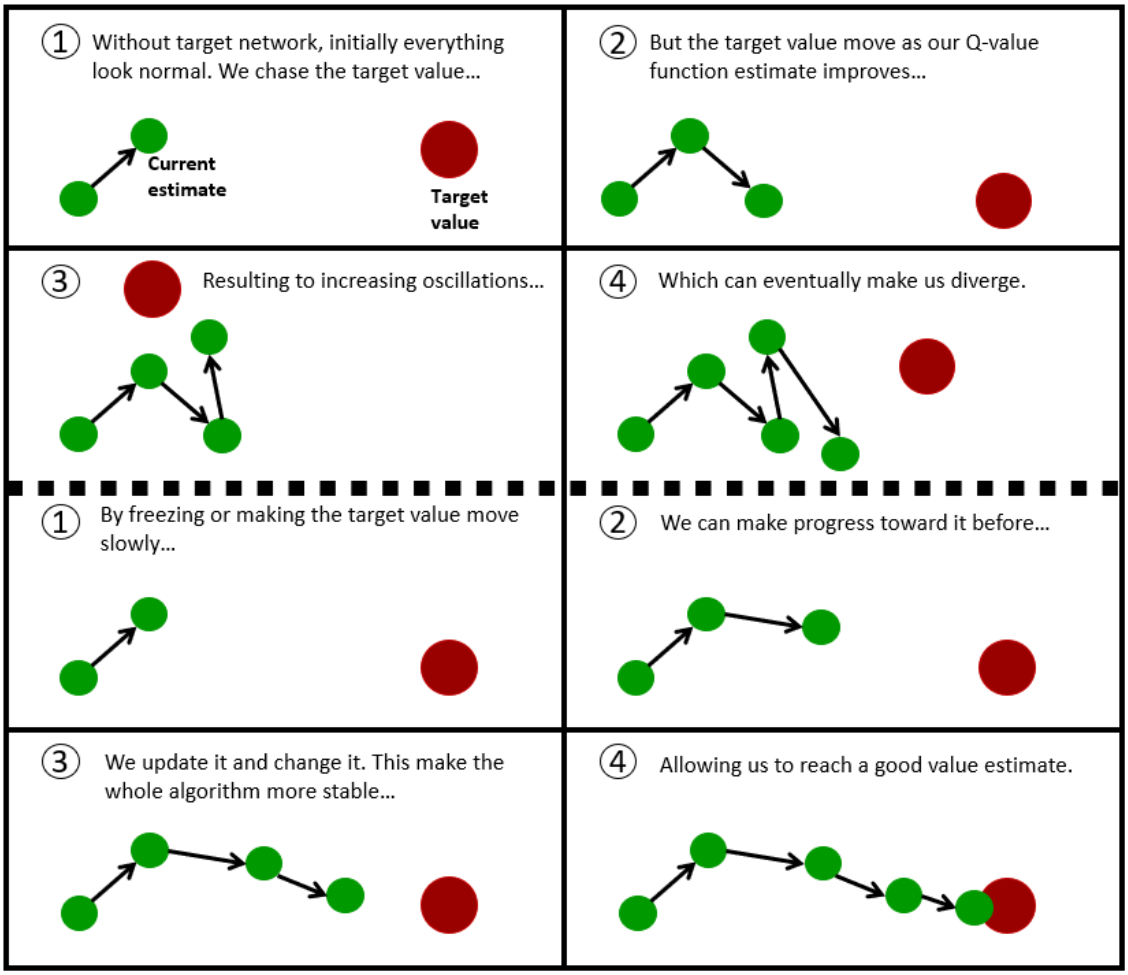


Figure 2.19: *Illustration of the moving target value problem from reinforcement learning.*

2.2.9 Limits

Deep reinforcement learning has been used intensively over the past few years. It is attracting a lot of attention from both academia and industry because it provides a framework to learn the solution to a rational decision-making problem, and most of today's problems of interest can essentially be described as such problems. Recent breakthroughs in deep neural networks make it, even more, attractive as no a priori expert knowledge is required anymore. Nevertheless, the deployment of DRL-based agents is still limited. Their applications seem to be restricted to games and their performance varies from one implementation to another. This is not only due to the stochastic nature of these algorithms, but more importantly because of their fundamental elements [Gho+21] making it hard to build an agent that generalizes well enough to be useful. In this section, we propose to discuss these principal components of DRL which are known to restrain the usage of these methods.

Reward design is hard

The performance of the Reinforcement Learning methods is directly related to the reward function. Some might even say that the reward function is, for better or worse, the *unique piece of intelligence* in the Reinforcement Learning framework [Sil+21]. Its importance has been studied for years [DL04; DK12] but the field is still lacking formal design rules. Writing a reward function is not hard, what is hard is designing a reward function that encourages the desired behavior while still being learnable. Unfortunately, shaped rewards can bias learning which can lead to behaviors that don't match what we want. To design a good reward function, we need to understand well the process and the environment because, for many reward functions, different behavior can generate the exact same output. Let's illustrate this with the following example.

Consider a vehicle whose objective is to reach an objective position of coordinates in a 2-dimensional space $O_P = [X_o, Y_o]_t$ from a starting position $S_P = [X_s, Y_s]_t$. In order to design a reward function to teach the vehicle to perform this task, we first can think of a metric to use as a cost function so as when minimized, the target is reached. An intuitive candidate is the Euclidean distance $D_{L2}(t)$ computed at timestep t between the vehicle position $V_P = [X_v, Y_v]_t$ and the target position as:

$$D_{L2}(t) = \sqrt{\sum_{i=1}^{i=dim(V_P)} e_i^2(t)}, \tag{2.87}$$

with $e_i = O_{P_i} - V_{P_i}$ the error between the vehicle and target position for the i-th dimension of the 2-dimensional reference frame.

We know that when this distance measure is equal to 0, it means that the vehicle and the target position are superposed in space, thus the objective is reached. A straightforward reward function would consist in designing a reward that is a function of this distance as:

$$r(s_t) = -D_{L2}(t). \tag{2.88}$$

With this reward function (2.88), the closer the vehicle is to the waypoint and the higher the reward will be. The resulting optimal policy will thus reduce the distance at each timestep until it is equal to 0, resulting in the success of the task. However, when taking a deeper look at this simple reward function as illustrated by the resulting spatial distribution of rewards in Figure 2.20, we can see in Figure 2.20 that different states can generate exactly the same reward.

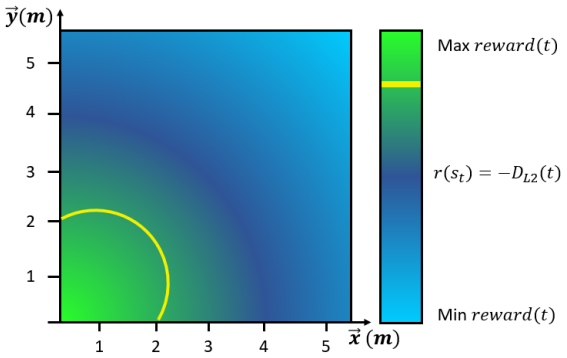


Figure 2.20: *Spatial distribution of rewards obtained with the reward defined in Eq.* (2.88).

In Figure 2.20 we represent the spatial distribution of rewards according to the reward function (2.88). The problem here is that because the reward is a function of the distance to the waypoint (as (2.88) is essentially a measure of the Euclidean distance) there is an infinite set of states that return the exact same reward value. This is illustrated by the continuous yellow line in Figure 2.20. In fact, each point on this continuous line is associated with the same Euclidean distance to the setpoint. Therefore, despite being intuitive at first, this reward function (2.88) can be problematic because very different trajectories can lead to the same value. This can make it very difficult for the Q-Value estimator to determine which trajectory is really the best for the long-term objective.

One way to tackle this problem is for example to consider the derivative of the distance $d_{rate} = D_{L2}(t-1) - D_{L2}(t)$ in the reward function as follows:

$$reward(t) = -D_{L2}(t) + d_{rate}. \tag{2.89}$$

With this new reward function (2.89), each time the actions reduce the distance between the vehicle and the target position, a positive reward is generated and add to the previous reward (2.88), otherwise a negative one is sent. By doing that, two points on the yellow line in Figure 2.20 will be associated with the same value if and only if they are associated with the same value of d_{rate} which is drastically less likely to be the case compared to with the first reward function Eq. (2.88). This will make it easier for the Q-Value estimator to differentiate trajectories from each other.

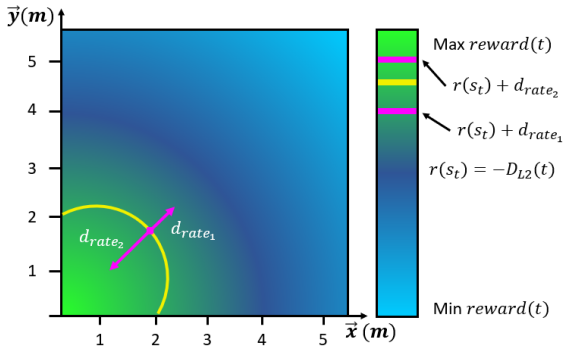


Figure 2.21: *Spatial distribution of rewards in the (x, y) plane obtained with the reward defined in Eq.* (2.89)*. The points that are at the same distance to the waypoint (i.e. yellow line) can now be associated with a different reward value based on the derivative of the distance.*

Depending on the controlled system, the process itself, the characteristics of the environment, or even the presence or not of reward signals, designing a suitable reward function can or not be feasible. There is not yet a uniform theory for the design of these objective functions. It remains an open question as to either if the reward is enough to drive behaviors that exhibits intelligent abilities [Sil+21] in spite of being the central piece of reinforcement learning algorithms. In our applications of AUVs adaptive control, a natural candidate for the reward function is the sum of errors on each DoF. We will see in this thesis that a proper reward design will be required in order to reach satisfying performance. The second component which limits the use of DRL is the Experience Replay mechanism. In fact, DRL uses the agent's past experience to improve its function approximators and we will see next how it can influence badly the performance of the resulting policy.

Experience matters

In order to improve itself, Policy Gradient methods are required to experience good trajectories. In accordance with the policy gradient (2.45), the policy improves solely if trajectories that are better (i.e. associated with a higher Q-Value) than the mean one are experienced. Therefore, using these samples in the maximum entropy policy update (2.77) is primordial. As presented in Section 2.2.6, the ER technique is the fundamental technique to choose those samples. Nevertheless, even with this standard formulation cf Section 2.2.6, there is a great number of parameters that are usually ignored despite having an impact on the learning performance, including:

- **The replay buffer size**: the total number of transitions that the replay buffer can store. When its maximum size is reached, the replay buffer is accessed in a first-in-first-out fashion. The bigger the replay buffer size, the more the data will look like it is IID, which in turn improves the gradient update quality. However, if the replay buffer is too big, an important transition will have much less chance of being used to update the policy, which could impair the learning process. In contrast, if the replay buffer is too small, the learned policy can be the result of an overfitting process on recent transitions, which precludes performance improvement.
- **The age of a transition:** the number of gradient steps taken by the agent since the transition was generated. This value can be seen as a measure of the extent to which the transitions stored in the Replay Buffer are off-policy, as it tells us how different the current policies are from those stored in the buffer. The age of the oldest policy stored increases with respect to the buffer size.
- **The replay ratio:** the number of gradient updates per environment transition. It can be viewed as a measure of the frequency at which the agent is learning using existing data versus learning from collecting new experiences.

The size of the replay buffers, however, can impact negatively the learning performances [ZS17]. There are two competing methods that can be used to solve this issue: the Combined ER (CER) [ZS17] and the Prioritized ER (PER) [Sch+16]. CER consists of adding the latest transition performed to the mini-batch pooled over the replay buffer, whereas with PER important transitions, as measured by their associated TD-error (2.73)(2.74), are given a higher probability to be used in the gradient updates. Using CER, however, the last transition will undoubtedly be sampled and instantly affect the policy.

Nevertheless, even with CER, a drop in performance was observed for certain sizes of replay buffer, at some point in the training (even when tuning the learning rate). This behavior was related to the process itself rather than to the aforementioned parameters [ZS17]. As written in [ZS17], “CER is a workaround …and future effort should focus on developing a new principled algorithm to fully replace ER.”

A recent detailed analysis of ER is provided in [Fed+20], where an analysis of the effects of the aforementioned parameters are presented. Several conclusions on how the parameters can affect the learning dynamics are drawn. These conclusions can be summarised as follows:

- Increasing the replay capacity while fixing the age of the oldest policy improves the performance because it lowers the chances of overfitting to a small subset of (state, actions).
- As the agent trains, it spends more time in higher quality regions of the environment (as measured by rewards), thus learning to better estimate the return in such regions leads to further gains in performance.
- Increasing the buffer size with a fixed replay ratio has varying improvements. The replay ratio stays constant when the buffer size is increased because of both the replay capacity and the age of the oldest policy increase. If one of these two factors is independently modulated, the replay ratio will change.

In practice, these elements may be difficult to control independently and may also have different effects on the data distribution itself depending on the precise architecture of the agent. These issues highlight the entanglement that exists between these different properties of the experience replay mechanism at the level of practical algorithmic adjustments and motivates further study into how these properties can be disentangled. This direction of research is particularly important with regard to obtaining agents which can effortlessly scale with increased availability of data. Finally, the combination of Bellman Equations with deep neural networks itself is problematic because it induces overestimation in value estimation. We present next how this problem arises and why it is inherent to DRL algorithms.

The distribution shift problem in RL

In deep learning, and in regression, in particular, we optimized function approximation so as to match the training data. The performance of the model is then evaluated on data that were not observed during training. In reinforcement learning, the model (i.e. the agent) is often trained in simulators that can in theory provide an infinite amount of data. The agent is then evaluated by interacting with another version of the environment (which can be explicitly a region of the environment that was not explored during training, another the same environment but with different disturbances or uncertainties, or even other tasks). As illustrated in Figure 2.22, there is no guarantee that the solution obtained from the training set will hold its performance on another set of states, that is the real world. This problem is also known as the distribution shit problem.

Let's now illustrate this problem mathematically. As proposed in [Kum+19], consider an empirical risk minimization problem where want to minimize some loss:

$$\theta \leftarrow \arg\min_{\theta} \mathbb{E}_{x\sim p(x), y\sim p(y|x)}[(f_\theta(x) - y)^2], \tag{2.90}$$

where we are essentially doing regression on a data distribution $f(x) \rightarrow y$, and the training samples are coming from $p(x)$, and the goal is to minimize the empirical risk. We can ask, given a new x^*, is my learned $f(x^*)$ going to be correct? This is a more tricky question that we can think of. What we can say is that if the model is not overfitting, then the expected value of the loss is low:

$$\begin{aligned} &\mathbb{E}_{x\sim p(x), y\sim p(y|x)}[(f_\theta(x) - y)^2] \text{ is low.} \\ &\mathbb{E}_{x\sim \hat{p}(x), y\sim p(y|x)}[(f_\theta(x) - y)^2] \text{ is not low, in general } \hat{p}(x) \neq p(x). \end{aligned} \tag{2.91}$$

What if x^* is sampled from $p(x)$ (i.e. $x^* \sim p(x)$)? Even there the error on x^* is not necessarily low, it is low in expectation, but x^* could be that one point where we got unlucky. Usually, we do not worry about this in deep learning, because we are using neural networks, and deep neural networks generalize well. This is true in most cases, including supervised and unsupervised learning. However, what if we do not just pick x^* randomly, what if we pick it to maximize $f_\theta(x)$:

$$x^* \leftarrow \arg\max_{x} f_\theta(x). \tag{2.92}$$

This would be very dangerous as illustrated in Figure 2.22.

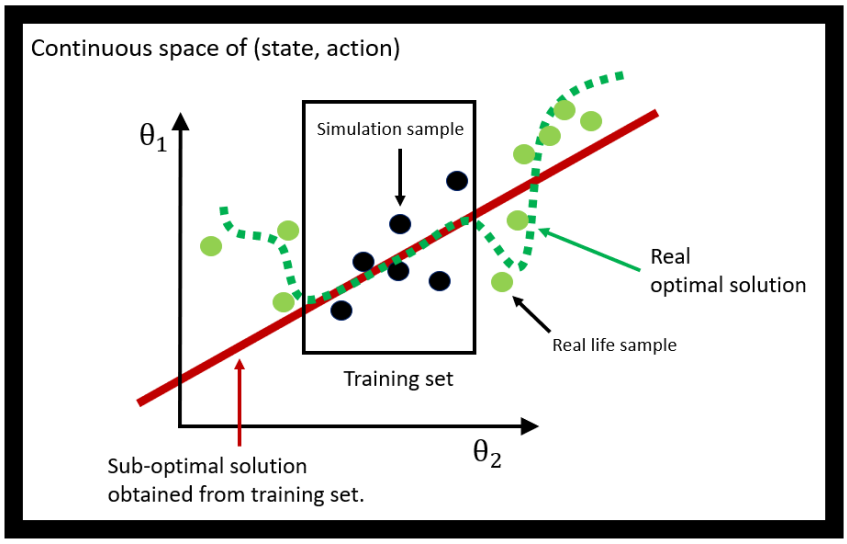


Figure 2.22: *Illustration of the distribution shift problem in reinforcement learning. There are no guarantees that the solution obtained from training, often using samples from a simulator, will remain valid on new state samples (e.g. real-life samples).*

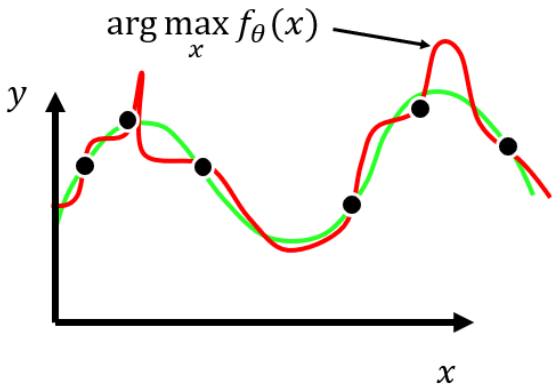


Figure 2.23: *Imagine the green curve is the true function and the red curve is our fit. While the red curve is good in most cases, when we pick the maximum value of the function, we might pool out exactly that point where $f(x)$ overestimates to a larger degree (i.e. the largest positive error). This is similar to an adversarial example, in a particular direction.*

As illustrated in Figure 2.23, even if we think that our model has generalized very well, if we pick x^* by maximizing our model, then we can always trick it toward a wrong value. This is very problematic, and it turns out that this maximization is exactly what we do in Q-Learning. For recall, the Bellman equation:

$$Q(s,a) \leftarrow r(s,a) + \max_{a'} Q(s',a'), \tag{2.93}$$

can be written differently as:

$$Q(s,a) \leftarrow r(s,a) + \mathbb{E}_{a' \sim \pi_{new}}[Q(s',a')], \tag{2.94}$$

where instead of writing it as a max, we can write it as an expected value under some distribution π_{new}, where π_{new} is that $\arg\max$ policy (making both equations equivalent). When we do Q-Learning, the right side of Eq. (2.94) is used as the target and the objective is to minimize the error on that target value:

$$\min_Q \mathbb{E}_{(s,a)\sim\pi_\beta(s,a)}[(Q(s,a) - y(s,a))^2] \tag{2.95}$$

We expect a good accuracy when $\pi_\beta(a|s) = \pi_{new}(a|s)$. But how often does that happen? The whole point of training is to find a π_{new} that improves over π_β, and even worse we are going to pick $\pi_{new} = \arg\max_x \mathbb{E}_{a\sim\pi(a|s)}[Q(s,a)]$ that is exactly that maximization depicted in Figure 2.23 that is so problematic because since we are using deep neural networks to approximate the Q-Value function, as we are only interested in its maximum value (which is associated to the optimal action according to Bellman Equations), there is a great probability that the error on the maximum of the estimate is very high. This is why we see the overestimation in Q-Learning [Kum+19] because essentially that maximization is finding these adversarial examples which produce large error values, and because we are iterating this, those errors accumulate more and more until we have massive Q-Value overestimation. This is basically the challenge of distribution shift in reinforcement learning because as overestimation can easily happen in the Q-Value framework, the resulting policy tends to overestimate its future gains in a new environment, which makes it very difficult to effectively transfer such agents in the real world.

In this section, we have discussed the different limits of DRL methods by analyzing various variables and parameters that directly affect the performance of the resulting policy. Another component that can greatly affect the performance of the agent is the quality of the training data. In fact, as DRL methods require a large amount of data to converge, they are traditionally trained under simulation and if not realistic or representative enough of the agent and its environment, DRL methods can also fail at learning at all. In the following, we will present the simulation tools we used in this thesis.

2.3 Simulation framework for deep reinforcement learning

2.3.1 ROS

In this thesis, the experimental platform are using mainly the Robot Operating System (ROS) framework [Qui09] for sensors communication and actuators control. ROS is the most commonly used open-source meta-operating system for robots. It comes with a large number of off-the-shelf tools, denoted as packages, developed by the community. ROS comes with a large number of off-the-shelf tools, denoted as packages, developed by the community. From computer vision to advanced control, these packages are designed in such a manner that only the high-level variables are required to reuse the packages, facilitating code and knowledge reuse. ROS has been used for many years now, both in academia and industry and because there exist a lot of great tutorial on how to use it, we will not present ROS in its totality.

2.3.2 Gazebo-based AUV simulation

The UUV Simulator package

Deep Policy Gradient methods need a large amount of interaction with the environment in order to converge to a satisfying behavior. Their initial behaviors are mostly random and thus risky for robotic platforms. For this reason, it is common practice to perform their training in a simulator at first, which in theory, can provide an infinite amount of training data. In this thesis, Gazebo [KH04] was chosen as the simulation environment for training our agents because it is based on ROS which is the main framework used on the robotic platforms considered for the experimental validation. For underwater environments simulation, we used the ROS package called UUV Simulator [Man+16]. It is a Gazebo-based library of AUVs and underwater environments allowing us to run personalized missions as illustrated in Figure 2.24 where an AUV faces an underwater wreck. The UUV Simulator [Man+16] can simulate several current and wave disturbances,

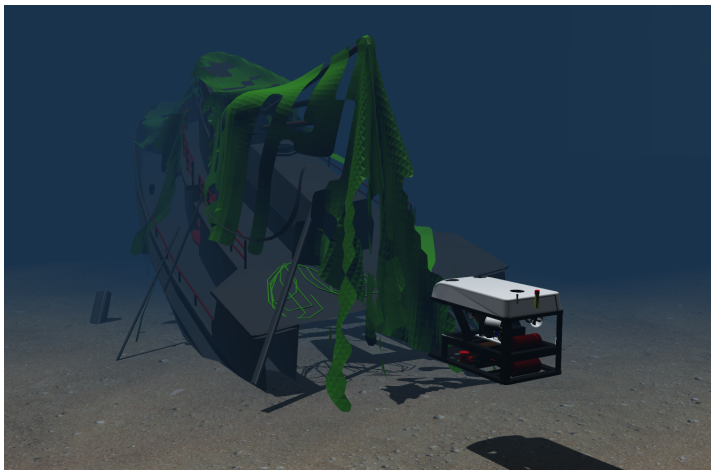


Figure 2.24: *The RexRov2 platform in a simulated Gazebo environment from UUV Simulator.*

thruster dynamics and body wrench disturbances. When incorporated into the simulations, the induced forces have a realistic physical impact on the robots and fluid dynamics. The sea current disturbance (which is the main focus of this study) is modeled as a uniform force acting over the Gazebo environment. This force is represented by a linear velocity, v_c (in $m.s^{-1}$), a horizontal h_c and a vertical angle j_c (measured in radians). These current variables can be changed at any step in the simulations through ROS-based callbacks or directly through Python/C++ scripts. The simulated RexROV2 platform is equipped with an IMU which feedbacks its linear velocities and orientation (Euler angles). These variables are accessible through ROS topics, which are essentially data pipelines to access the simulation variables. Our software architecture consists in using the simulation meta-data to train the learning algorithms considered in this work. To that end, we fix the simulation's real-time factor at 1, thus the training time is equivalent to what could be experienced on a physical platform.

In order for RL algorithms to learn efficiently, the data inside the state vector should allow it to find the correlation between the actions and the environment changes: the difference between successive states should be large enough. This can be ensured by setting what we define as *Sampling rate* which is the rate at which a state vector is sampled from the environment after the execution of an action. A good practice consists in synchronizing it with the slowest sensor of the system which ensures the fastest sampling rate with no potential loss of information in the state vector (since at least the slowest sensor will be updated, it ensures that each variable of the state vector has changed since the last sampling).

The Rexrov2 platform

The RexROV2 platform provided in UUV Simulator [Man+16] is based on the ROV Minerva platform. Complete modeling of this vehicle (including its kinematic, kinetic, and thruster model as well as its control allocation) can be found in [Ber12]. The RexROV2 can be modeled using the general equations of motion for a marine craft, which can be written in the vectorial form according to [Fos94] as:

$$\begin{aligned} \dot{\eta} &= J_{\Theta}(\eta)\nu, \\ M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) &= \delta + \delta_{cable}, \end{aligned} \tag{2.96}$$

where η and ν are the position and velocity vectors respectively, δ is the control force vector and δ_{cable} is the vector describing the umbilical forces from the cable attached to the ROV. The RexROV2 is an ROV-type platform provided in UUV Simulator. It is propelled by 6 thrusters (complete details on its equation of motions are provided in [Ber12; McC16; Yan+15]. The control vector u is obtained by transforming the actuator force vector:

$$\delta = \mathbf{T}(\alpha)\mathbf{K}\mathbf{u}, \tag{2.97}$$

where $\mathbf{T}(\alpha) \in \mathbb{R}^{n \times r}$ is the thrust allocation matrix; $\mathbf{K}$ is the thrust coefficient matrix; δ is the control force vector in n DOF and $\mathbf{u} \in \mathbb{R}^r$ is the actuator input vector. UUV Simulator can be used to derive a vector of thruster contributions for every DOFs (for clarity, $\sin(\cdot)$ and $\cos(\cdot)$ are denoted as $s\cdot$ and $c\cdot$ below):

$$\mathbf{T}_i = \begin{bmatrix} \textit{Surge} \\ \textit{Sway} \\ \textit{Heave} \\ \textit{Roll} \\ \textit{Pitch} \\ \textit{Yaw} \end{bmatrix} = \begin{bmatrix} c\theta c\phi \\ s\theta c\phi \\ s\phi \\ -Zs\theta + Ys\phi \\ -Zc\theta + Xs\phi \\ -Yc\theta + Xs\theta \end{bmatrix}. \tag{2.98}$$

These vectors are then assembled into a thrust allocation matrix $\mathbf{T} = [T_1, \ldots, T_6]$ which describes the relationship between propeller thrust and the vehicle speed, [Car18]. Using this allocation matrix, the control inputs u are transformed as $u = [v_x; v_y; v_z; \omega_\psi; \omega_\theta; \omega_\phi]^T$ where $[v_x; v_y; v_z]^T$ are linear velocity inputs (in $m.s^{-1}$) and $[\omega_\psi; \omega_\theta; \omega_\phi]^T$ are torque inputs (in radians) expressed in the reference frame attached to the center of mass of the simulated RexROV2 platform.

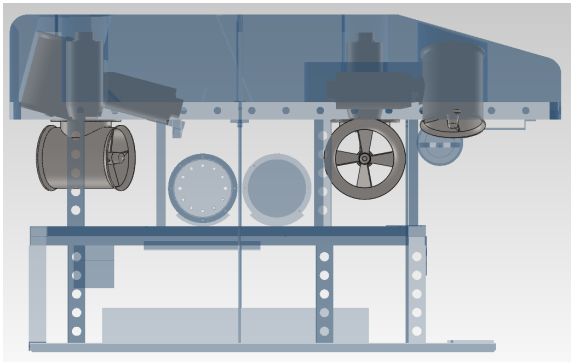


Figure 2.25: *RexROV2 thruster positions, side view. Source: [Ber12].*

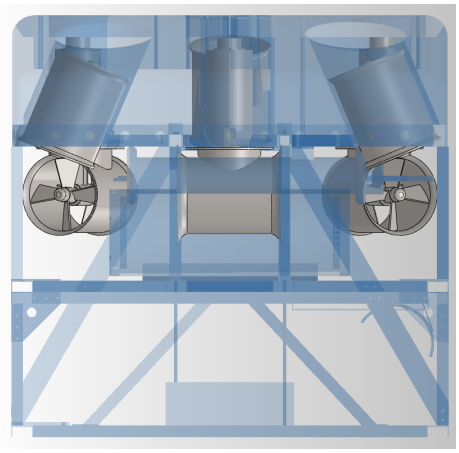


Figure 2.26: *RexROV2 thruster positions, front view. Source: [Ber12].*

The Bluerov2 platform

As shown in Figure 2.27, the Blue Robotics BlueROV2 Heavy platform is a small-size ROV that is mostly used for observation applications. It includes four horizontal and four vertical thrusters of type T200 that are placed as illustrated in Figure 2.27. Similarly to the RexRov2 platform, the Bluerov vehicle can be modelled according to [Fos94] as:

$$\begin{aligned} \dot{\eta} &= J_{\Theta}(\eta)\nu, \\ M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) &= \delta + \delta_{cable}, \end{aligned} \tag{2.99}$$

where the above parameters have been described in Section 2.3.2. Exhaustive modeling of the vehicle will not be provided in this thesis as the complete description of the vehicle was previously performed by our team from Flinders University in [WS18]. Nevertheless, some assumptions have to be made on the vehicle in order to be able to consider the proposed control designs:

- The Bluerov operates at relatively low speeds (i.e. less than 2 $m.s^{-1}$), thus the lift forces can be neglected.
- The Bluerov is assumed to have port-startboard symmetry and fore-after symmetry. The center of gravity (CoG) is therefore assumed to be located in the symmetry planes.
- The Bluerov is assumed to be hydrodynamically symmetrical about 6-DoFs. Thus, the motions between the DoFs of the vehicle in hydrodynamic can be decoupled.

Again, we have access to a thruster allocation matrix that allows us to control the Bluerov in the 6 degrees of freedom (Surge, Sway, Heave, Roll, Pitch, and Yaw). A complete description of the vehicle and its control parameterization is proposed in [WS18].

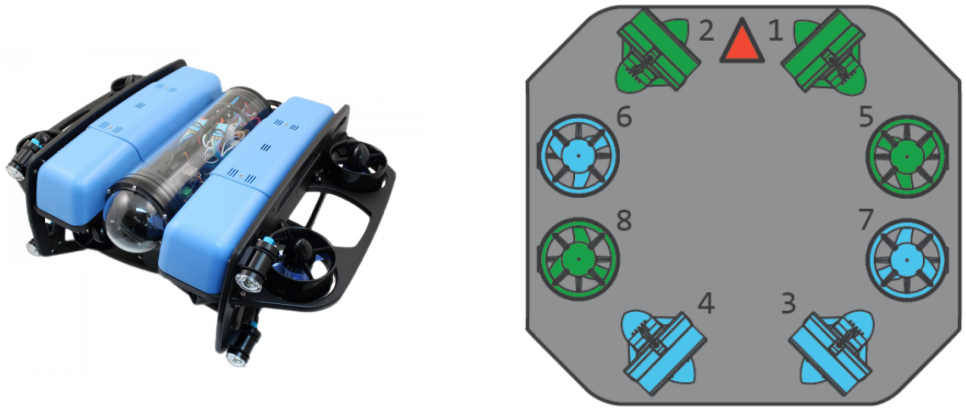


Figure 2.27: *Illustration of the Bluerov configuration kit (left) and The Bluerov thruster configuration from top-down view. Green and blue thrusters indicate counter-clockwise propellers respectively (right). Source: BlueRobotics.*

2.4 Summary

In this section, the technical background elements necessary to design the proposed learning-based adaptive control system have been presented. First, the elements on adaptive control theory were described with a focus on the AUV applications. We proposed a classification of solution methods based on their dependence on the process model which leads to three classes of adaptive control methods: model-based, model-free, and learning-based. We have provided a description of each of these methods and we have shown why a learning-based adaptive controller is the best choice for the problem considered in this thesis which is AUV control under unobservable current disturbance. We discussed why reinforcement learning is relevant here to design such a control system and the limits of adaptive control theory are provided at the end of the associated section.

Then, the technical background elements of reinforcement have been presented. We proposed this time classification of solutions methods based on the nature of the decision-making process which lead to three classes of reinforcement learning methods: model-based, value-based, and policy gradient methods. We provided a description of each of these methods and we outlined why deep policy gradient methods are the more relevant choice to adjust the control parameters. The limits of reinforcement learning theory are discussed at the end of the associated section.

Finally, the simulation tools and the vehicle modelization is provided with a description of the ROS-based package that we used to simulate AUV applications. The general block diagram of the desired learning-based adaptive controller is illustrated in Figure 2.28. Simulated data are used to optimize a policy whose objective is to estimate the best control parameters for a given state of the process. This policy is learned using Off-Policy TD Learning as depicted by the presence of the Experience Replay in the off-line learning loop. During online control, the control structure adapts its parameters based on process feedback.

In the next section, we described the proposed novel learning-based adaptive controller, in order to fill the gaps in Figure 2.28, along with all the preliminary studies that lead to its design.

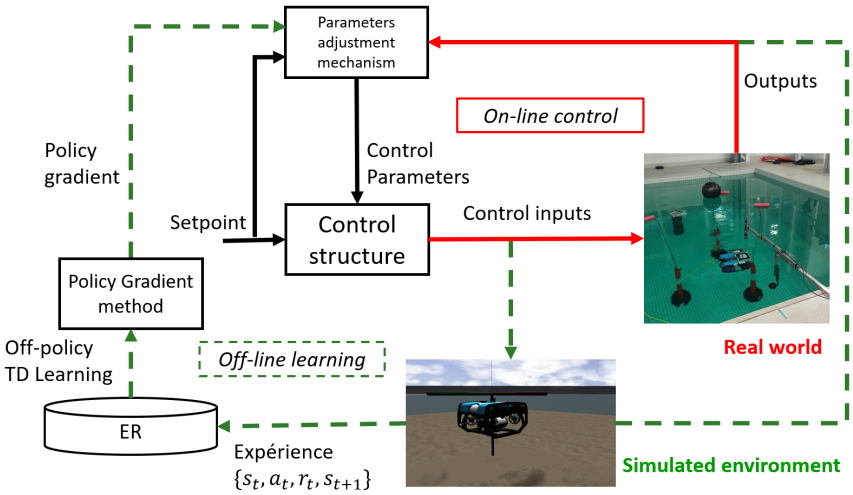


Figure 2.28: *General block diagram of a DRL-based learning-based adaptive control system.*

3 Proposals for a novel learning-based adaptive control system

The exercise of going through a Ph.D. is a particular acting exercise where one person is trying to focus on one problem, that it does think is interesting enough, and that will be able to make a reasonable amount of advance to it in a given time frame. In practice, it turns out very rapidly that the problem of interest has already been studied for decades, or even longer, and ideas that were once believed to be groundbreaking turn out to have been already disproved experimentally. From this return to reality, a reasonable first step is to study the related works of the community to identify what seems to be achievable and where contributions can be made. In this section, we present this procedure with first an analysis of related works in AUVs learning-based adaptive control. Then, based on the resulting findings, we present the preliminary studies that lead to the final design of the proposed learning-based adaptive control system.

3.1 Related works in AUVs learning-based adaptive control

The Deep Deterministic Policy Gradient algorithm (DDPG) [Lil+16] was used in [Yu+17] to learn the optimal trajectory tracking control of AUVs. This control problem consists in keeping the error $e = x - x_d$ between the actual trajectory x and the target x_d at zero. The authors concentrate on trajectory tracking in the (x, y) plan. The ideal trajectory is defined as the Euclidean distance to the desired path. The error to be minimized is measured as the Euclidean distance between the current and the desired trajectory. The DDPG architecture is composed of critic and actor neural networks. In this architecture, the actor chooses which action to take and the critic tells it how good this choice was. The actor parameters are then updated using this information to improve its future decision-making processes. The authors defined a loss function to update the parameters of the actor-network which include Lyapunov stability components [BR01]. They proved that reaching the minimum value of the loss results in the optimal desired behavior, making the error converge to zero in a stable fashion (with respect to Lyapunov stability theory). The performance of the proposed approach was tested on straight-line and curved trajectories, showing that the DDPG algorithm was able to effectively solve the task in both cases. This approach was compared to a fixed gain PID and the results indicate that the learning-based controller exhibits better performance in terms of tracking error. However, as the stability components are incorporated in an indirect way (i.e. by an additional term in the actor loss function) there are no formal guarantees that the system will remain stable at all times.

In [Wan+18], the DDPG algorithm was used to learn adaptive trajectory planning for multiple AUVs in under-ice environments. This work also considered an additional objective in terms of satisfying constraints related to kinematics, communication range, and sensing area. This is a challenging environment, where there is great uncertainty related to the current flow under the ice. The water temperature as well as partially submerged ice structures (against which acoustic signals bounce) can substantially impact the sensor feedback, directly influencing the control performance. In order to solve this challenge, the authors proposed a new cost function together with an experience replay mechanism. The cost function used to update the Policy network includes terms that represent the field uncertainty, the cost of the trajectory, and the constraints induced by the trajectory. By making the reward a function of the control constraints, the associated optimal behavior policy takes into account these restrictions during the decision-making process. The traditional experience replay technique is also modified to store past experiences of the agent that specifically satisfy the communication range and sensing area constraints in distinct replay buffers. The gradient updates are then applied using transitions from these replay buffers only. This procedure helps the agent to identify positive actions (with respect to these constraints) more rapidly since the associated behaviors will have a much higher probability to be selected during gradient descent. This resulted in a robust behavior policy that chooses actions that comply with the aforementioned constraints. Simulation results showed that this approach was able to achieve a performance matching that of a benchmark method which assumes perfect knowledge of the field hyper-parameters. Again, the stability components were incorporated in an indirect fashion.

Learning-based adaptive control was investigated in [KNS19] for the station keeping of an AUV under unknown currents. They used the DDPG algorithm to control the position of a BlueROV2 platform in surge x and sway y combined to a PD control law that regulated the AUV position in heave z and orientation in roll ϕ, pitch θ and yaw ψ. The DRL algorithm was used [KNS19] to learn a PD control law as a function of the vehicle position and velocity at previous time steps. The training was performed within the ROS-based Gazebo simulator [Qui09]. The evaluation was conducted on a real platform in an indoor water tank. The authors stated that the agent's performance was satisfactory after 600 episodes. The resulting behavior policy was then used on a real platform to solve various tasks including a DP 4-corner test scenario, which is a benchmark test for validating dynamic positioning (DP). It was observed that the agent is able to complete

the station-keeping task even when facing this unknown operating condition. The experimental evaluation consisted of three scenarios: two different desired pose definitions and a 4-corner test. The first scenario consists in changing one error state while in the second scenario, both error states are changed at the same time. The 4-corner test consists in performing station keeping at the 4 corners of a rectangular trajectory. These experiments proved that the agent is able to complete the task under real conditions. The performance was, however, slightly worse in the real environment compared to the simulated one, especially for the most challenging task of a DP 4-corner test.

More recently, Deep Imitation Learning (DIL) [Liu+18; Pen+18] and another Deep Policy Gradient algorithm named Twin Delay Deep Deterministic Policy Gradient (known as TD3) [FHM18], were combined in [Chu+20] for the design of a learning-based controller for an AUV (the combination of DIL and DRL is denoted as DIRL). The goal of this work was to leverage imitation learning (IL) to speed up the training of DRL methods such as DDPG in order to facilitate the sim-to-real transfer of RL-based agents. The idea of IL is to use some expert agent to generate examples of appropriate behaviors that are then used to perform the pre-training procedure of the ANNs (in a supervised fashion). Then, the networks can be fine-tuned using the normal DRL framework under a reduced number of episodes. Depending on the task, expert knowledge can be provided by a human operator (remotely controlling the robotic platform). This knowledge can also be obtained by using some *Robust* or *Optimal* control designs (derived from model-based theory using the *a priori* knowledge on the process and plant) to regulate the controlled system. In both cases the essence of the work is the same, that is to build a data set out of experiences generated by an expert agent and to use it for prior offline learning. Here, the PID structure was also considered but this time as the expert agent [Chu+20]. The PID controller is used to construct a set of state s_t and thrust vectors F_t as expert demonstration data. The overall training procedure is defined as follows: until the replay buffer reaches a specific size, the training is performed on the expert data ((s_t, F_t) from the PID controller) in a supervised scheme; when the replay buffer size exceeds this threshold, the Policy network is further trained in an RL procedure with the TD3 algorithm. Their method is hence denoted as IL-TD3. Two control tasks were considered: 1) constant depth and attitude control and 2) depth trajectory tracking control. For the first task, a total of 400 episodes were performed and it was observed that after only 100 episodes of supervised learning, the behavior cloning was almost complete. In addition, the authors trained the TD3 and DDPG algorithms for the same problem (i.e. with the same loss functions). They showed that IL-TD3 is the fastest to converge and the most stable during training. They compared IL-TD3 under simulation to the original PID controller with and without current disturbances. Results showed that, in the case of no disturbances, both methods were able to solve the task (IL-TD3 exhibited faster response and lower overshoot but at the cost of a much higher thrusters solicitation than the PID algorithm). In the case of current disturbances, the PID controller failed at solving the task. Thanks to prior expert knowledge, the IL-TD3 was able to capture actions associated with high-value rewards despite the early stage of the training. This, according to the authors, leads to further gains. The advantage of their method was moreover demonstrated with real-life tank experiments on the BlueROV2 platform. The Policy network, which was trained under simulation only, was able to provide satisfying control ability when transferred in the real world.

This initial literature study allowed us to identify the advantages and drawbacks of different designs of such control systems. We were able to observe that two trends are dominating the field of adaptive control of AUVs: direct and indirect approaches. In the first case, the parameters of the controller are adjusted directly using some Machine Learning techniques. In the second case, the adjusted control parameters are the result of an optimization problem where the state and/or unknown parameters of the process are estimated and then used to compute the associated optimal parameters. In most cases, these approaches are applied to classic model-based control structures such as the PD or PID control laws. The objective is then to adjust the parameters of these control structures, namely their gains, according to process variation and using deep reinforcement learning. The principal RL algorithms used in the literature are the TD3 and the DDPG algorithms. These deep policy gradient methods (See section 2.2.7) build deterministic actors and do not take into account the entropy term from the maximum entropy reinforcement learning framework (presented in Section 2.2.8). Most of these works use the original experience replay mechanism detailed in Section 2.2.6 except [Wan+18] where they proposed to select the past experience of the agent based on different control constraints and to store them in different replay buffers accordingly. By using only these particular samples to update the actor, the resulting policy displays a more robust behavior with respect to these constraints. Based on these findings, the next step of the thesis consisted in performing a number of preliminary studies on the design of RL-based control systems for AUVs with the objective of tackling the above-mentioned challenges.

3.2 Preliminary studies

In this section, we present some preliminary studies that guided the design of the learning-based adaptive controller proposed in this thesis.

3.2.1 Model-based vs model-free adaptive control of AUV under current disturbance

Following the analysis of the related works presented in the previous Section 3.1, the next step of this thesis consisted in evaluating the first design of an adaptive controller based on deep reinforcement learning. In collaboration with Yoann SOLA, another Ph.D. student at ENSTA Bretagne, we proposed an end-to-end model-free adaptive controller based on the SAC algorithm. The purpose of this study was to compare this approach to a purely model-based PID controller and the results were presented [Sol+20b] at the IEEE Global Oceans Conference 2020. In the following, we summarize the main results of this paper, and its preprint version can be found in Appendix C.

Task description

In this study [Sol+20b], we address the control problem of target rallying by an AUV. The goal is to stabilize the vehicle at a given position and bearing angle. The state of the vehicle at the time step t denoted as $\mathbf{x}_v(t)$ is defined by its Cartesian position and Euler orientation $\mathbf{x}_v(t) = [x_v\ y_v\ z_v\ \psi_v\ \theta_v\ \phi_v]^T$ (respectively roll, pitch and yaw for its orientation). The target is defined as $\mathbf{x}_w(t) = [x_w\ y_w\ z_w\ \phi_w]^T$, and the values of these variables are provided by the Gazebo simulator as described in Section 2.24. We defined o_w the bearing angle between the vehicle and the target as:

$$o_w = \begin{cases} atan2(y_w - y_v, x_w - x_v) - 2\pi & \textbf{if}(\phi_w - \phi_v) > \pi \\ atan2(y_w - y_v, x_w - x_v) + 2\pi & \textbf{else.} \end{cases} \tag{3.1}$$

The task of target rallying can be achieved if the distance measure to the target is minimized:

$$d_t = \sqrt{(x_v - x_w)^2 + (y_v - y_w)^2 + (z_v - z_w)^2} \leq d_{reached}, \tag{3.2}$$

where d_t is the Euclidean distance measure, and $d_{reached}$ is the threshold value that we want the distance signal d_t to be lower than. This class of control objective is used in various AUV missions, such as autonomous docking or underwater inspection, where a conservative regulation of the AUV position is required.

Design of the model-based controller

We used as evaluation platform the RexROV2 vehicle provided by UUV Simulator presented in Section 2.3.2. Using the thruster allocation matrix from [Man+16], we can directly control the vehicle in the surge, sway, heave, roll, pitch and yaw as described in Section 2.3.2. Thus, we can use a PID-type control law (i.e. model-based) to control each of these DoFs to perform the target rallying mission. The PID control input is defined as:

$$u_i(t) = k_p e_i(t) + k_i \int e_i(t) + k_d \dot{e_i}(t). \tag{3.3}$$

The model-based controller consists in using the PID control law for all 6 DoFs with the target value for the roll and pitch being 0 and for the yaw angle being the bearing angle o_w defined in Eq. (3.1). In order to evaluate the performance benefits of the proposed approach, we use a fixed but optimal version of the PID controller (3.3). The parameters of the controller, namely its gains k_p, k_i, and k_d, have been optimized and are fixed during operation. Thus, this controller will be denoted in the following as a PID controller. The gains of the resulting PID controller (provided by the UUV Simulator) were tuned using the model-based optimization method called SMAC [HHL11]. Details on the gains can be found on the UUV Simulator website and the complete list of the controller parameters are provided below in Table 3.1.

Gains	Surge	Sway	Heave	Roll	Pitch	Yaw
Kp	11993.888	11993.888	11993.888	19460.069	19460.069	19460.069
Ki	321.417	321.417	321.417	2096.951	2096.951	2096.951
Kd	9077.459	9077.459	9077.459	18880.925	18880.925	18880.925

Table 3.1: *The gains obtained from the model-based optimization scheme.*

Design of the model-free controller

As presented in Section 2.1.2, in model-free control we aim at maximizing an objective function without any prior information on the system and process models. For this reason, we propose a model-free controller which consists in using Deep Reinforcement Learning to learn how to control the vehicle by applying control inputs T directly on the vehicle thrusters, thus without the thrust allocation matrix presented in Section 2.3.2. The objective of the learning agent is to build a stochastic predictive model, using the SAC algorithm, that maps the thrusters inputs (2.97) directly from the current state:

$$\begin{cases} \pi_\theta : \ S \subset \mathbb{R}^{20} & \rightarrow \ A \subset \mathbb{R}^{12} \\ \qquad x = [\mathbf{s}_t]^T & \mapsto \ [\lambda_i, \mu_i]\,. \end{cases} \tag{3.4}$$

Thus, the outputs of the Policy network are the 6 pairs of (λ_i, μ_i). Since we are building a stochastic actor, the control inputs applied to the i-th thruster of the AUV are denoted as $T_i \in [-240, +240]$ and are modeled by a Gaussian distribution $\mathcal{N}_i(T_i)$ defined as:

$$\mathcal{N}(T_i) = (2\pi\mu_i)^{-1/2} \exp\Big\{ -\frac{1}{2\mu_i}(x - \lambda_i)^2 \Big\}, \tag{3.5}$$

where $\lambda_i \in \mathbb{R}$ and $\mu_i \in \mathbb{R}^+$ are the mean and variance of $\mathcal{N}(T_i)$ that are estimated by the Policy network. In the following, this controller will be denoted as an RL controller.

State vector

Since the proposed approach here only focuses on the high-level guidance and low-level control parts of a GNC system for an AUV, the navigation considerations are not tackled. Therefore we consider that the AUV has access to good estimates of its pose, its linear and angular velocities, and its tracking errors with respect to the waypoint. The environment state $\mathbf{s}_t$ observed by the SAC algorithm at time t is given as follows:

$$\mathbf{s}_t = [\ \mathbf{x}\ ;\ \Theta;\ \mathbf{v}\ ;\ \Omega\ ;\ \phi_e\ ;\ \mathbf{x}_e\ ;\ \mathbf{u}_{t-1}\ ]^T, \tag{3.6}$$

where

- $\mathbf{x} = [\ x,\ y,\ z\ ]$ is the position vector of the AUV in Cartesian coordinates,
- $\Theta = [\ \psi,\ \theta,\ \phi\ ]$ is its orientation vector expressed with Euler angles (respectively roll, pitch and yaw),
- $\mathbf{v}$ is its linear velocity vector (the temporal derivative of $\mathbf{x}$),
- Ω is the angular velocity vector (the temporal derivative of Θ),
- ϕ_e is the error between its current yaw and the desired yaw leading directly towards the waypoint,
- $\mathbf{x}_e$ is the error between its current position and the desired position, corresponding to the waypoint position,
- and $\mathbf{u}_{t-1}$ is the vector of the inputs of the actuators computed at the previous step, at time $t-1$.

Reward function

In order to perform a waypoint tracking mission, we designed the following reward function r_t (3.7). It was inspired partly by [Car+18], where the reward function takes into account low-level variables such as linear and angular velocities and their respective references. We adapted this work for a higher level of control, taking directly into account the position of the AUV and its reference. Therefore the guidance and control parts of the GNC system of the AUV are both provided by the SAC algorithm.

$$r_t = \begin{cases} r_{toward} & \textbf{if } \ d_t - d_{t-1} > 0 \\ r_{backward} & \textbf{else} \\ r_{waypoint} & \textbf{if } \ d_t < \epsilon \\ r_{limit} & \textbf{if } \ z \notin [z_{min}, z_{max}] \end{cases} \tag{3.7}$$

where r_t is the reward received by the agent at time t, d_t is the current relative distance between the AUV and the waypoint to reach, z_{min} and z_{max} are the authorized limits for the vertical movement z of the AUV, and ϵ is a strictly positive real number.

Each term appearing in (3.7) represents a specific feature of the global desired behavior of the AUV:

- r_{toward} is a variable reward given when the distance d_t is decreasing, which means that the AUV moves toward the waypoint. It is defined as follows:

$$r_{toward} \;=\; \lambda_1(d_t - d_{t-1}) \;-\; \lambda_2 \,\|\Omega\|, \tag{3.8}$$

  where λ_1 and λ_2 are positive weighting terms. The term weighted by λ_1 rewards large movements toward the waypoint, while the term weighted by λ_2 penalizes strong angular speeds, and promotes indirectly a softer use of the actuators.

- $r_{backward}$ is a constant negative reward given when the distance d_t is increasing, which means that the AUV moves backward the waypoint.

- $r_{waypoint}$ is a constant positive reward given to the agent when the AUV reaches the waypoint, which leads to a terminal state, ending the current episode.

- r_{limit} is a constant negative reward given to the agent when the vertical movement of the AUV exceeds the limits defined by $[z_{min}, z_{max}]$, which leads to a terminal state, ending the current episode.

We chose the value of all the parameters in order to give the signals r_{toward} and $r_{backward}$ a magnitude of around 10, as recommended in [Haa+18c]. The complete list of hyperparameters used to design the DRL controller is provided in Table 3.2. To make the training more realistic, and to avoid overfitting, we will describe next how we randomize several parameters of the environment during training.

Training

The training of the RL controller consists in performing a total of 1200 training episodes. This amount of episodes represents approximately 4 hours of real-time training, after which convergence in terms of success rate was reached (i.e. the variance of rewards was small enough, for us, to consider the policy sufficiently good to stop the training). A training episode is defined as follows: at the beginning of the episode, we initialize the vehicle at a position $\mathbf{x} = [0, 0, -20]$ (in meters and relative to the frame attached to the Gazebo world center) with a random orientation Θ, where $\psi = \theta = 0$ and $\phi \in [0; 360]$ (in degrees and with respect to the center of mass of the vehicle body). The waypoint is then placed at a random position located at a maximum euclidean distance of 50 meters from the AUV initial position whose coordinates are used as a setpoint. Then, the RL controller is launched and the episode ends either when the target is reached, when the vehicle depth exceeds a predefined threshold, or when the maximum episodic step size is reached.

The sensors measurements included in our state vector incorporate added noise such as: for each $x_i \in \mathbf{s}$, $x_i = x_i + \sigma_i$ with σ_i randomly sample from the uniform distribution $\mathcal{U}[0.05; 0.1]$, except for the past actions where σ_i is there randomly sample from the uniform distribution $\mathcal{U}[0.01; 0.05]$ (because the amplitude of the actions is notably smaller than the other variables). We also added fluctuating sea currents to our underwater simulated environment. The current velocity $c_v \in [0; 1]$ (in $m.s^{-1}$) and angles, $(c_{ha}; c_{va}) \in [-0.5; 0.5]$ (for horizontal and vertical angles respectively in $radians$), are randomly modified every 100 time steps during training and evaluation.

Table 3.2: *List of hyperparameters and their values.*

Training hyperparameter	Value
SAC version	1 (see Section 2.2.8)
Activation function	Leaky ReLU
Optimizer (all networks)	Adam [KB15]
Learning rate (all networks)	3×10^{-4}
Discount factor (γ)	0.99
Mini-batch size	256
Target network smoothing coefficient (Δ) 2.2.8	0.005
Update frequency (all networks)	1
Layer Normalization [BKH16] (all networks)	True
Replay buffer max size	$1e6$
Replay start size	$1e4$
Experience Replay method	Original ER [Lin04]

Evaluation

The evaluation consists of using both controllers for a series of 500 simulated episodes each with the same environment parameters, including: AUV initial position, target positions, and sea currents characteristics. We measure their performance in terms of success rate (percentage of episodes where the AUV reaches the waypoint), temporal mean, and standard deviation (SD) of the distance error $d\delta$ to the ideal trajectory (the direct path from the initial position of the AUV to the waypoint location), but also in terms of thrusters usage under the form of the temporal mean of the Euclidean norm of the input vector **u** at each step (with each element of **u** being a signal between -240 and $+240$).

Table 3.3: *Task performance.*

Performance criterion	MB Controller	DRL Controller
Success rate	96%	86%
Mean $d\delta$ (m)	3.81	8.67
SD of $d\delta$ (m)	3.53	5.45
Mean of $\sum \|\|\mathbf{u}\|\|$	541.42	481.06

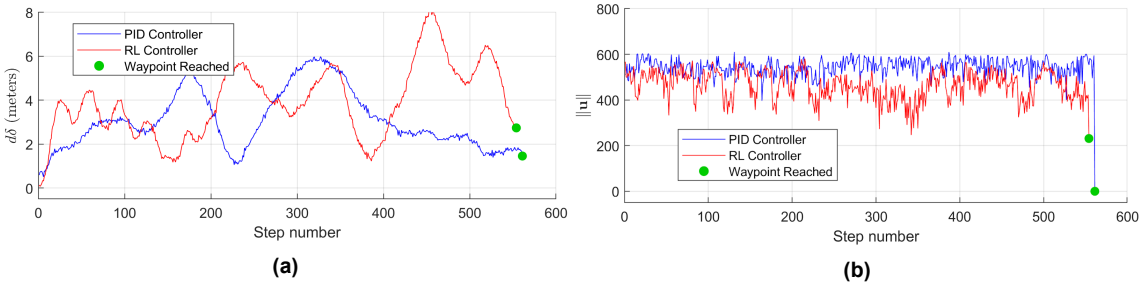


Figure 3.1: *In (a) we plot the distance error $d\delta$ from the ideal trajectory over time steps. and in (b) we display the Euclidean norm of the input vector **u** over time steps for the associated episode.*

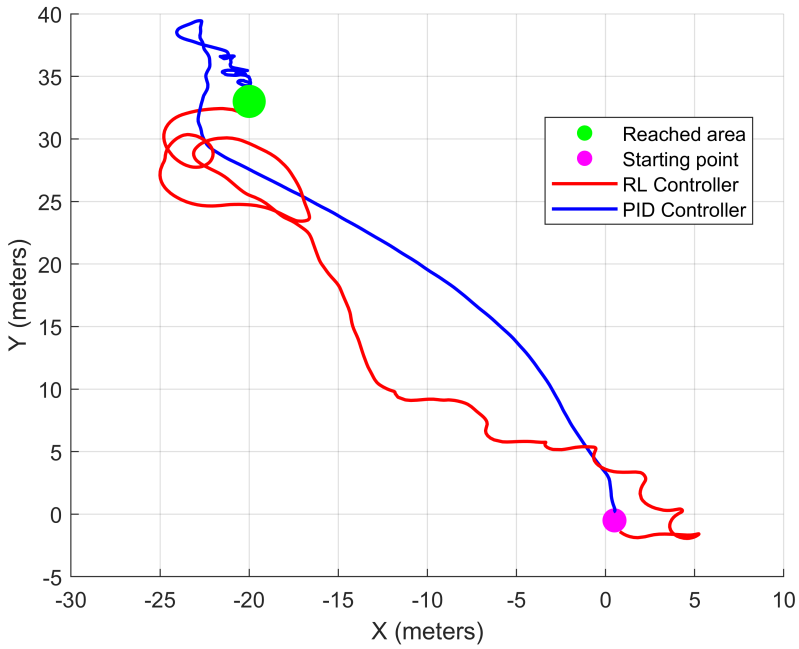


Figure 3.2: *Trajectory in the XY-plan performed during an episode by both controllers with $C_v = +0.096\ m.s^{-}1$, $C_{ha} = -0.1324\ rad$ and $C_{va} = 0.083\ rad$.*

Discussion

In table 3.3 we provide the evaluation results with the success rate, the mean distance to the target, the standard deviation of the distance to the target, and the mean value of the sum of control inputs. We can see that despite being capable of completing the task, both controllers display very different behaviors as illustrated in Figure 3.2 where we plot the 2D trajectory performed by each controller for the same target and sea current disturbance. In terms of success rate, the PID controller is doing better than the RL controller with 10% more success. When taking a look at the mean and standard deviation of the distance to the optimal trajectory (that is the line passing through the starting point and the target), we can also observe that the PID controller is doing better in both metrics. This trend is illustrated in Figure 3.1a where we plot the evolution of the euclidean distance to the target for the same episode characteristics. We believe that this is due to the model-based part that is included in the PID controller. In fact, with the PID control law Eq. (3.3), as described in Section 1.1, the closed-loop control is continuously monitoring the error value based on feedback measurements (i.e. model-based information). Therefore, in order to reduce the error on a given DoF, the PID controller knows, through the thruster allocation matrix, which combination of thrusters to use in order to regulate that particular DoF.

We can see that the RL controller has lower thruster usage, and thus indirectly in power consumption. This is illustrated in Figure 3.1b where we plot the evolution of the norm of the sum of the control input vector, again for the same episode. The difference in power consumption can be explained by the characteristics of the controllers. In fact, the PID controller incorporates the correlation between the DOFs of the RexROV 2 platform, whereas the RL controller does not. This means that to move in a specific direction, the PID controller knows exactly which combination of thrusters to use. The RL controller, on the other hand, estimates directly the inputs to the actuators and tends to use fewer thrusters than the PID controller to perform the same movement. This leads to smaller power consumption but poorer control performances than the PID controller.

Eventually, we can observe that despite reaching the goal, the trajectory of both controllers is not close to the optimal one. This is due to the current disturbance that is not implicitly included in both control methods: the PID control law takes into account only the state of the vehicle (its position and bearing angle) while the RL controller does not include the characteristics of the current disturbance in the state vector Eq. (3.6) because in practice we do not have a sensor to measure it. However, we saw that the closed-loop feedback controller is able to compensate for disturbances by means of adaption (see Sections 1.1). It sounds intuitive that we could benefit from both paradigms by using Deep Reinforcement Learning to learn how to adapt the parameters of the closed-loop feedback controller with what is denoted as a learning-based adaptive controller (see Section 2.1.2). Instead of estimating directly the control inputs to apply to the thruster, our idea is to use deep reinforcement learning to adjust the parameter of the model-based control structure (e.g. the PID control law) in a model-free manner. In the following, we present a second preliminary study that is our first attempt at designing such a control system. This work was in collaboration with ONERA, the French National Aerospace Laboratory, as part of my MSc. graduation internship performed in that institution on the use of deep reinforcement learning for the control of a terrestrial vehicle [Cha+20b].

3.2.2 Model-free vs learning-based adaptive control of MAV under wind disturbance

Following the work described in Section 3.2.1, and in accordance with previous work in collaboration with ONERA [Cha+20b], we proposed a first design of a learning-based adaptive controller using deep reinforcement learning. The purpose of this study was to compare this approach to a purely model-based and a purely model-free counterpart of the same control structure in order to identify the benefits of the proposed approach. The objective is to observe the effect of merging together model-based and model-free theories on both the control and learning performance. Despite the main focus of this thesis being AUVs, we apply this methodology on an aerial drone as it is the principal system of interest of ONERA. Later, we will present how to improve and adapt this method for an AUV. The results were published [Cha+22] as a chapter of the Lecture Notes in Electrical Engineering book series (LNEE, volume 793). In the following, we summarize the main results of this paper, and its preprint version can be found in Appendix C, with the related work of learning-based adaptive control in the aerial domain. We performed this study under simulation with Gazebo again and by using the ROS package called RotorS [Fur+16] described in Appendix B.

Task description

In this study [Cha+22], we address the same control problem of target rallying as described in Section 3.2.1 but this time using a Micro Aerial Vehicle (MAV). The full description of the MAV and its simulation using Gazebo is provided in the paper in Appendix C. The state of the vehicle at the time step t denoted as $\mathbf{x}_v$ is defined by its Cartesian position and Euler orientation $\mathbf{x}_v = [x_v\ y_v\ z_v\ \psi_v\ \theta_v\ \phi_v]^T$ (respectively roll, pitch and yaw for its orientation). The target is defined as $\mathbf{x}_w = [x_w\ y_w\ z_w]^T$, and the values of these variables are provided by the simulator Gazebo. Similarly, the control objective is to minimize the Euclidean distance d_t between the MAV and the target (see Section 3.2.1 for all the details on the error signals).

Design of the model-free adaptive controller

Compared to the related work presented in the paper [Cha+22] available in Appendix C, we propose to only treat the parameters adjustment task. We will now present the design of a model-free adaptive control strategy initially proposed in the first preliminary study (see Section 3.2.1) but this time for the application of a target rallying mission by a MAV under unknown wind gusts. The methodology follows the same line of thought as the previous study [Sol+20b]. The objective of the learning agent is to build a predictive model, using the SAC algorithm, that directly maps the hexacopter control inputs defined in B. In fact, with the RotorS package [Fur+16], the MAV can be controlled in terms of vertical thrust force, and roll and pitch orientation. Therefore, the model-free adaptive controller consists in building a predictive model that maps a state vector directly to the MAV control inputs:

$$\begin{cases} \pi_\theta :\ S \subset \mathbb{R}^{114} & \rightarrow\ A \subset \mathbb{R}^6 \\ \qquad x = [\mathbf{s}_t]^T & \mapsto\ [\lambda_i, \mu_i]\,. \end{cases} \tag{3.9}$$

With the model-free adaptive controller, the outputs of the Policy network are the 3 pairs (λ_i, μ_i) of mean and standard deviation. The control inputs $[T_{\phi_\mathrm{r}}, T_{\theta_\mathrm{r}}, T_\mathcal{T}]$ applied to the MAV is modeled by Normal distributions defined as:

$$\mathcal{N}(T_i) = (2\pi\mu_i)^{-1/2} \exp\{-\frac{1}{2\mu_i}(x-\lambda_i)^2\}, \tag{3.10}$$

where $(T_{\phi_\mathrm{r}}, T_{\theta_\mathrm{r}}) \in [-\frac{\pi}{6}; +\frac{\pi}{6}]$ are roll and pitch angle inputs, and $T_\mathcal{T} \in [m\times g; m\times(g+3.0)]$ (with $m = 1.544$Kg, $g = 9.81\mathrm{m.s}^{-1}$) is the vertical thrust force input that is sampled as described in Section 3.2.1. For summary, the outputs of the Policy network are 3 pairs of (λ_i, μ_i), and the control inputs are then directly sampled from the Gaussian distributions $\mathcal{N}_i(\lambda_i, \mu_i)$ and applied to the MAV.

State vector

In order to achieve the target rallying mission, we need to provide relevant data to the agent. Therefore, we defined the following vector o_t as the observation at the time-step t of the environment:

$$o_t = [\ a_{t-1}\ ;\ v_x\ ;\ v_y\ ;\ v_z\ ;\ \omega_\phi\ ;\ \omega_\theta\ ;\ \omega_\psi\ ;\ \mathbf{x}_v\ ;\ e_t\ ;\ d_t\ ] \tag{3.11}$$

where

- a_{t-1} are the last actions performed,
- $[v_x; v_y; v_z]$ and $[\omega_\phi; \omega_\theta; \omega_\psi]$ are the MAV linear and angular velocities,
- $\mathbf{x}_v \in \mathbb{R}^6$ represents its position and orientation,
- $e_t = [e_x; e_y; e_z]$ are the current errors on the target in terms of Euclidean distance,
- and d_t is the current Euclidean distance between the hexacopter and the target.

All the variables involved in the observation vector Eq. (3.11) are assumed to be measured, their estimation is out of the scope of this work (as they are provided by the Gazebo simulator). The dimension of this observation vector Eq. (3.11) is 19 and it has been standardized to have zero mean and a variance of 1. In order to provide a higher time horizon to the agent, we constructed the state vector out of the current and past two observations vectors $[o_t; o_{t-1}; o_{t-2}]$. For the purpose of providing the agent a sense of "velocity" in the evolution of the state, we consider the two-by-two difference of these vectors such as $vel_t = (o_t - o_{t-1})$ and $vel_{t-1} = (o_{t-1} - o_{t-2})$. We went even further and tried to add a sense of "acceleration" in the evolution of the state by including the difference between the latter vectors $acc_t = (vel_t - vel_{t-1})$. The resulting state vector is therefore defined as:

$$s_t = [\ o_t\ ;\ o_{t-1}\ ;\ o_{t-2}\ ;\ vel_t\ ;\ vel_{t-1}\ ;\ acc_t\ ] \tag{3.12}$$

The dimension of the state vector Eq. (3.12) is thus 114. Here, we synchronized the sampling rate (defined in Section 2.3.2) with the embedded odometry sensor which led to a *sampling rate* of about 20Hz. This means that every time a new sensor message is received from the topic associated with the odometry sensor, a new state vector is captured and the next action is sampled.

Design of a learning-based adaptive controller

As stated in Section 2.1.2, learning-based methods consist in exploiting standard model-based control architectures that are either tuned or redesigned by a model-free algorithm in order to compensate for the unknown part of the model. The MAV is subject to an additive but unknown wind perturbation which can be modeled as:

$$u_{adp} = u + u_{\text{wind}} \tag{3.13}$$

In this context, despite u_{wind} being unknown, the PID control law can again be considered as the integral term that will ensure convergence to the steady state despite the wind disturbance. In order to derive the PID control law, let's defined here the control objective in more detail as stabilizing the MAV at a given target in space $\mathbf{x}_w = [x_w y_w z_w]$ with a velocity $v = \dot{\mathbf{x}}_w$. Defining the error vector as $e = \mathbf{x}_v - \mathbf{x}_w$, the wind disturbance is taken into consideration by considering an additional steady-state variable $z = \int_0^t e(\tau)d\tau$. The augmented model with state vector $X = [z, e, v]$ becomes a double integrator:

$$\begin{bmatrix} \dot{z} \\ \dot{e} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} z \\ e \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \tag{3.14}$$

The corresponding PID control law is defined as:

$$u = -k_i z - k_p e - k_d v \tag{3.15}$$

The poles of the closed-loop system are solutions to the following equation:

$$\lambda^3 + \lambda^2 k_d + \lambda k_p + k_i = 0 \tag{3.16}$$

We propose to re-parametrize the PID control law to make it adaptive using pole placement. This way, the action space for learning purposes is limited to desired solutions in the real part of the pole map, which prevents sampling unnecessary solutions in the space of the control gains. The desired constants $\tau_1 > 0, \tau_2 > 0, \tau_3 > 0$ are then defined as:

$$\lambda_1 = \frac{-1}{\tau_1}\ ;\ \lambda_2 = \frac{-1}{\tau_2}\ ;\ \lambda_3 = -\frac{1}{\tau_3} \tag{3.17}$$

Since each one is solution to (3.16), it follows that:

$$\begin{bmatrix} 1 & \frac{-1}{\tau_1} & \frac{1}{\tau_1^2} \\ 1 & \frac{-1}{\tau_2} & \frac{1}{\tau_2^2} \\ 1 & \frac{-1}{\tau_3} & \frac{1}{\tau_3^2} \end{bmatrix} \begin{bmatrix} k_i \\ k_p \\ k_d \end{bmatrix} = \begin{bmatrix} \frac{1}{\tau_1^3} \\ \frac{1}{\tau_2^3} \\ \frac{1}{\tau_3^3} \end{bmatrix} \Leftrightarrow MK^T = N \tag{3.18}$$

Finally, the gains of the controller (3.15) are obtained as $K^T = M^{-1}N$:

$$\begin{aligned} k_i &= \frac{1}{\tau_1\tau_2\tau_3}, \\ k_p &= \frac{\tau_1 + \tau_2 + \tau_3}{\tau_1\tau_2\tau_3}, \\ k_d &= \frac{\tau_1\tau_2 + \tau_1\tau_3 + \tau_2\tau_3}{\tau_1\tau_2\tau_3}, \end{aligned} \tag{3.19}$$

where $\tau_i \in \mathbb{R}^+$.

We proposed to use the SAC algorithm to estimate at each time step the best values of poles Eq. (3.16) that can be mapped into gains values according to Eq. (3.19) to derive the resulting control input. For this first design of a learning-based adaptive controller, we proposed to use the incremental pole adjustment method which consists in adding a small value to the poles starting from a model-based configuration of pole values that will be named in the following as nominal pole values. By doing so, we avoid jumping from a configuration of gains to a totally different one, which can give rise to undesired oscillations here as this first mapping Eq. (3.19) is not bounded. The nominal set of pole values is obtained using the empirical Ziegler-Nichols method in an environment without wind disturbance. The resulting poles for each DoF are then equal to:

$$\tau_1 = 1,\ \tau_2 = 2.5,\ \tau_3 = 0.875. \tag{3.20}$$

which is, when using our mapping proposed in Eq. (3.19), equivalent in the space of gains to:

$$k_p = 2,\ k_i = 0.457,\ k_d = 2.542. \tag{3.21}$$

This nominal set of pole values is not optimal because it was determined for one operating condition only which in addition did not incorporate wind disturbance (because it was too difficult to make the MAV converge under wind disturbance without a good enough set of gain values). Therefore, our approach consists in starting from this nominal set (3.20) and updating the pole values at each time step using $\tau_i(t+1) = \tau_i(t) + \nabla_i$ with $\nabla_i \in [-0.01; +0.01]$. Then, the updated gains are used to compute the associated PID control input that is ultimately applied. This is denoted as learning-based adaptive control because the resulting control input Eq. (3.15) is a function of model-based information (i.e. the measured error values) and of the outputs of a deep neural network (i.e. the adjusted pole values). The objective of the learning agent is thus to build a predictive model that directly maps ∇_i from the current state to adjust the poles starting from this initial configuration:

$$\begin{cases} \pi_\theta :\ S \subset \mathbb{R}^{222} & \rightarrow\ A \subset \mathbb{R}^{18} \\ \qquad x = [\mathbf{s}_t]^T & \mapsto\ [\lambda_i, \mu_i]\,. \end{cases} \tag{3.22}$$

With the learning-based adaptive controller, the outputs of the Policy network are the 9 pairs (λ_i, μ_i) of mean and standard deviation. The ∇_i added to the pole τ_i is modeled by Normal distributions defined as:

$$\mathcal{N}(\nabla_i) = (2\pi\mu_i)^{-1/2} \exp\{-\frac{1}{2\mu_i}(x - \lambda_i)^2\}. \tag{3.23}$$

In summary, with this controller the outputs of the Policy network are 9 pairs (λ_i, μ_i) of mean and standard deviation that are used to model normal distributions $\mathcal{N}_i(\lambda_i, \mu_i)$. The value ∇_i is then sampled from $\mathcal{N}_i(\cdot)$. The pole $\tau_i(t)$ is updated with $\tau_i(t+1) = \tau_i(t) + \nabla_i(t)$. Finally, the resulting poles are transformed back into the space of gains and the PID control law is computed and applied.

State vector

We used a slightly different observation vector from the model-free scheme (3.11). Indeed, we propose to add the resulting pole values and the PID controller outputs (3.43) in the observation vector:

$$\begin{aligned} o_t = [\ & a_{t-1}\ ;\ \tau_{rpt}\ ;\ pid_{rpt}\ ;\ k_{p_{roll}}\ ;\ k_{i_{roll}}\ ;\ k_{d_{roll}}\ k_{p_{pitch}}\ ;\ k_{i_{pitch}}\ ;\ k_{d_{pitch}}\ ;\ k_{p_{thrust}}\ ; \\ & k_{i_{thrust}}\ ;\ k_{d_{thrust}}\ ;\ \phi\ ;\ \theta\ ;\ \psi\ ;\ v_x\ ;\ v_y\ ;\ v_z\ ;\ \omega_\phi\ ;\ \omega_\theta\ ;\ \omega_\psi\ ;\ \zeta_t\ ;\ e_t\ ;\ d_t\ ] \end{aligned} \tag{3.24}$$

where $dim(\tau_{rpt}) = 9$, $pid_{rpt} = [\phi_{\mathrm{r}}\ ;\ \theta_{\mathrm{r}}\ ;\ \mathcal{T}]$ and $dim(o_t) = 37$. We constructed the state vector out of this observation vector $\mathbf{s}_t$ as earlier Eq. (3.12), resulting in a state vector of dimension 222.

Reward function

The following reward function r_t has been designed in order to teach the agent how to complete the mission of target rallying:

$$r_t = \begin{cases} r_{receded} \ \textbf{if}\ \ d_{rate} \leqslant 0, \\ r_{forward} \ \textbf{if}\ \ d_{rate} > 0, \\ r_{reached} \ \textbf{if}\ \ d_t \leqslant d_{reached}, \\ r_{failed} \quad \textbf{if}\ \ z_w \notin [0.25; 20], \end{cases} \tag{3.25}$$

where r_t is the reward of the agent at time step t; d_{rate} is the distance rate to the target performed between the last two time steps such as $d_{rate} = d_t - d_{t-1}$; $d_{reached}$ is the limit value beneath which we consider the target to be reached ; $z_w \in \mathcal{R}_W$ is the MAV altitude; both $r_{reached}$ and r_{failed} are terminal rewards determined at the end of the ongoing episode. Each of these terms represents the specific features of the desired behavior of the MAV:

- $r_{receded}$ is a constant negative reward equal to -20 that is sent to the agent whenever the robot is getting away from the target (or staying immobile).

- $r_{forward}$ is a positive reward generated when the relative distance to the target is decreasing as:

$$r_{forward} = C_1 \times e^{\left(-[(\frac{d_t}{1+d_{rate}})\times\frac{1}{C_2}]^2\right)}$$

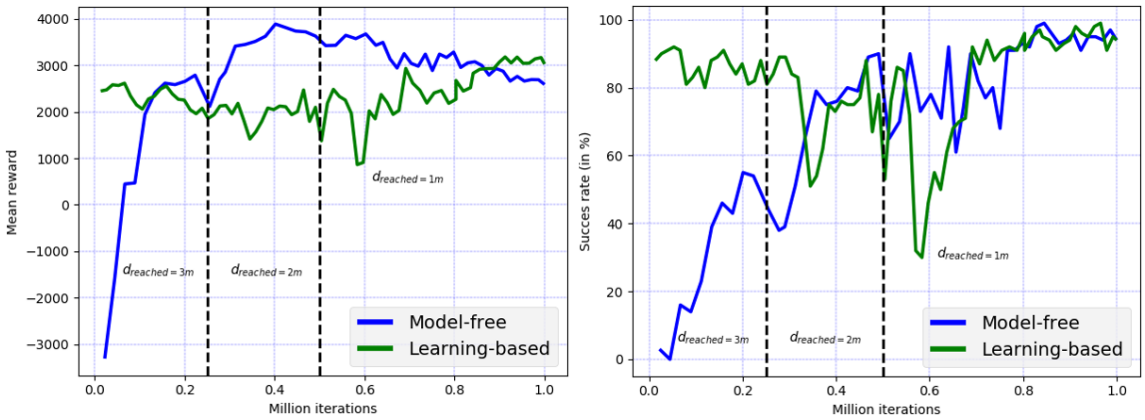
With this design, we encourage the robot to move toward the target as fast as possible. We chose the value of the constants C_1 in order to scale the positive reward signal. This is particularly important because as mentioned in [Haa+18c], the SAC algorithm is particularly sensitive to the scaling of the reward signal which is the magnitude of the reward value. We followed the recommendation prescribed in [Haa+18c] and chose to set $C_1 = 20$ in order to obtain a positive reward scale of 20 (which we found in practice to be the reward scale that gave us the best performance for this problem). The constant C_2 represents how sparse is $r_{forward}$ based on the distance to the target. We chose the value $C_2 = 20$ empirically. Therefore, the positive reward is equal to:

$$r_{forward} = 20 \times e^{\left(-[(\frac{d_t}{1+d_{rate}})\times\frac{1}{20}]^2\right)} \tag{3.26}$$

- A constant positive reward is sent to the agent when it succeeded to complete the mission, meaning $d_t \leq d_{reached}$. This generates $r_{reached} = +1000$.
- If the MAV altitude z_w exceed a threshold, the constant negative reward $r_{failed} = -550$ is generated.

Training

A training episode is defined as follows: at the beginning of the episode, the MAV is set at the center of the environment at an altitude of 3 meters with roll, pitch and yaw angles equal to 0. A target is then initialized at a fixed and uniformly random position $\Lambda = [\Lambda_x; \Lambda_y; \Lambda_z]^T$ with $[\Lambda_x; \Lambda_y]^T \in [-20; -5[\cup]5; 20]$ and $\Lambda_z \in [2; 20]$. The mission then begins and is considered a success if the relative distance to the target is inferior to a predefined threshold $d_{reached}$ and as a failure if an error signal is generated, both cases ending the episode. Otherwise, the episode is ended if the number of iteration steps reaches the maximum value allowed per episode which is set at 300. The training for both controllers consisted in performing $1\ 000\ 000$ iterations in an environment with a varying wind field (as described in Section B). To help the agent, $d_{reached}$ is reduced during training as follows: at first $d_{reached} = 3$m, from iteration $250\ 000$th we set $d_{reached} = 2$m and from iteration $500\ 000$th we set $d_{reached} = 1$m. The PyTorch framework [Pas+19] was used to carry out the numerical experiments, along with the CUDA toolkit [Nic+08] and an RTX 2060 GPU card, allowing us to perform the training of one controller in approximately 10 hours. It can be seen in Figure 3.3 that during training, both the learning-based (LB) and model-free (MF) controllers were able to reach a high success rate under unknown wind gust disturbances, which shows the applicability of the SAC DRL procedure for this type of aerial navigation problems. The LB strategy presents a much higher convergence speed to a significant success rate than the MF strategy, which shows the great potential of combining model-based classical controllers with learning procedures. This can be explained by the model-based part of the LB controller which allows it to choose relatively good actions despite being at the early stage of the training session. Therefore, from the beginning of the training, the LB controller is able to explore a much higher part of the reward space than the one of the MF controller. We believe this significantly helps the DRL algorithm to find better overall strategies.



(a) *Mean reward* **(b)** *Success rate*

Figure 3.3: *Training curves showing the mean reward and success rate computed per 100 episodes over a moving window of 100 episodes.*

Table 3.4: *List of hyperparameters and their values.*

Training hyperparameter	Value
SAC version	1 (see Section 2.2.8)
Activation function	Leaky ReLU
Optimizer (all networks)	Adam [KB15]
Learning rate (all networks)	3×10^{-4}
Discount factor (γ)	0.99
Mini-batch size	256
Target network smoothing coefficient (Δ)	0.005 (see Section 2.2.8)
Update frequency (all networks)	1
Layer Normalization [BKH16] (all networks)	True
Reward scale	40
Automatic temperature adjustment	False
Replay buffer max size	$1e6$
Replay start size	$1e4$
Experience Replay method	CER (see Section 3.3.3)

Evaluation

The evaluation consisted in performing the target rallying mission in areas of the same environment that had never been explored by either controller during training (i.e. the wind field in these areas was totally unknown to the neural networks) with $d_{reached} = 1$m. The evaluation is composed of a total of 500 episodes, different from each other in terms of target position and with a max step size per episode of 1000. The targets during the evaluation were uniformly distributed in the space defined by $\Lambda = [\Lambda_x; \Lambda_y; \Lambda_z]^T$ with $[\Lambda_x; \Lambda_y]^T \in [-50; -20[\cup]20; 50]$ and $\Lambda_z \in [2; 20]$. The same set of evaluation episodes was used for each controller. We also evaluated a fixed control strategy which consisted of a PID controller with the fixed nominal poles configuration (3.20). In the following, this model-based optimal but nonadaptive controller will be denoted as OFP for Optimal Fixed Poles controller.

Table 3.5: *Control performance.*

Controller type	Mean step number	Mean total reward	Mean reward per step
OFP	488	710.797	1.454
Model-free	357	2238.970	6.256
Learning-based	**281**	**4034.434**	**14.346**

Table 3.6: *Task performance.*

Controller type	Success rate	Positive reward rate
OFP	50.6%	61.197%
Model-free	74.2%	86.056%
Learning-based	**91.6%**	**89.2%**

Discussion

The outcomes of this evaluation are provided in Tables 3.5 and 3.6. In terms of control performance, the learning-based controller is on average completing the task in 281 control steps. The Model-free and OFP controllers take on average 357 and 488 control steps respectively. In other words, the proposed learning-based controller is between 27% and 73% faster than the other methods (respectively for the Model-free and OFP controllers). In terms of reward, the learning-based controller is outperforming the other methods with a mean total reward between 1.80 and 5.68 times higher than the other methods (respectively for the Model-free and OFP controllers). The highest reward per step is obtained with the learning-based controller where it is between 2.29 and 9.86 times higher than the other methods (respectively for the Model-free and OFP controllers).

The learning-based controller is also dominating in terms of task performance. It displays a success rate of 91.6% against 74.2% and 50.6% respectively for the Model-free and OFP controllers. The positive reward rate represents the rate of actions that resulted in the reduction of the distance to the target over an episode length. This variable tells us how stable the convergence to the steady state is. We can see that despite using neural networks, both the Model-free and Learning-based controllers display a similar positive reward rate that is between 1.40 and 1.45 times higher than the OFP controller. This particular result shows the benefits of adapting the controller response using deep reinforcement learning as with the OFP controller, the poles are fixed and do not vary over the episode.

Furthermore, on average, fewer actions are required to achieve the task with the LB controller despite sharing the same *sampling rate* (see Section 2.3.2). The mean reward per step of the LB controller is more than 2 times higher than the Model-free one. On the other hand, the OFP controller is showing a much lower success rate, close to 50%. We observed that with this strategy, failures mostly consisted in cases where the MAV is close to the steady state but is being deviated by the wind gust. The vehicle is then not able to recover from this disturbance because the fixed poles are not conservative enough to provide satisfying performance over a large spectrum of wind characteristics. These first results validated the proposed learning-based adaptive control design. By keeping a model-based control structure (i.e. PID control law), we can use deep reinforcement learning to design an adaptive pole-placement procedure where the controller parameters are a function of the process state. Despite not taking into account the current disturbance, it is still implicitly observed by the adjustment mechanism (i.e. neural network) as the value of the vehicle velocity and acceleration are included in the neural network input vector.

Nevertheless, this first design Eq. (3.16) holds the Routh–Hurwitz stability criterion that by considering only pole values of null imaginary component and negative real component, it ensures the stability of linear time-invariant systems. Here, stability refers to the fact that the system output is bounded, which means that as time goes to infinity, the system will converge to a steady state. This stability component is not enough for real-world systems, where the great number of disturbances and uncertainties make the system time-variant, and thus the Routh–Hurwitz stability criterion does not hold anymore. In this case, the stability of the control system can still be ensured using the classic Lyapunov stability theory [Lib05]. In this field, systems are modeled from an energy point of view, and the stability of the system is assessed in terms of energy dissipation. In other words, if we can have the mathematical proof (using Lyapunov stability theory) that the energy of a system is always reducing over time, we can guarantee that the system is stable and will reach a steady state. This approach has been studied and used for decades and there are various tools already existing to perform such stability analysis on AUVs. However, as soon as a nonlinear function approximator, namely neural networks, are present in the control system, it makes the classic Lyapunov stability analysis difficult or even impossible. For this reason, and in accordance with the internship of Hector Kohler in our laboratory at ENSTA Bretagne, we proposed a methodology to assess the Lyapunov stability of a learning-based adaptive controller which we present in the next Section 3.2.3.

3.2.3 Lyapunov stability of learning-based adaptive control

In the case of model-based adaptive control, the stability analysis of AUVs control systems is straightforward to conduct and successful study can be found in the literature [Gon+21][WSS21][SSB18]. However, as previously discussed, in Section 3.2.2, the Lyapunov stability analysis [Lib05] is difficult to apply to control systems that use deep neural networks because we can not predict beforehand all the possible values that these nonlinear function approximators can output. If we want to design a stable AUV control system, one designer can restrict the space of control parameter possible values to a Lyapunov-based space using the numerous Lyapunov function candidates from the literature [Fos94]. However, by restricting such a way the space of control parameters, we might lose all the benefits of using deep reinforcement learning to adjust their values. In other words, if there are only a few desired possible values, there might not have an advantage to adjusting them. Moreover, the resulting Lyapunov-based space of control parameters does not take into account the process variation and therefore is not optimal over the entire operating regime. For this reason, we propose to analyse empirically the stability component of an AUV learning-based adaptive control system. The choice of algorithms and methodology was made by Hector Kohler who is the first author of the resulting paper, by the application remains the one of interest of this thesis which is the adaptive control of AUV. The results were summarized and published [Koh+22] and presented at the 14th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles (CAMS) 2022. In the following, we summarize the main results of this paper, and its preprint version can be found in Appendix C.

Task Description

In this study [Koh+22], we address the same target rallying mission with again the RexRov2 platform using UUV Simulator (see Section 2.3.2). In the following, we will use a slightly different notation compared to the one from the paper [Koh+22] in order to match the notations of this thesis. The tracking error e is defined as the error between, the current AUV's position and orientation, and a fixed target represented by a desired position and orientation. Thus, the control objective consists in minimizing the error between the vehicle state $\mathbf{x}_v$ and the desired setpoint $\mathbf{x}_w$:

$$e = \mathbf{x}_v - \mathbf{x}_w. \tag{3.27}$$

where the state of the vehicle η_d is its position and orientation:

$$\mathbf{x}_v = [x_v \ ; \ y_v \ ; \ z_v \ ; \ \psi_v \ ; \ \theta_v \ ; \ \phi_v], \tag{3.28}$$

and the desired setpoint is defined as:

$$\mathbf{x}_w = [0, 0, 0, 0, 0, 0]. \tag{3.29}$$

Stability analysis methodology

Learning-based adaptive control methods, where neural networks are used, are dominating in modern robotic applications. However, we can observe different dynamics in maritime applications where such methods are yet to be successfully deployed on real platforms and for long operations. This can be explained by the fact that in underwater applications as studied in this thesis, we have limited observability of the process due to the limited embedded sensors. In addition, these vehicles have to face various disturbances acting on their body that we are yet able to model properly. In this context, learning-based methods can hardly be considered as little to no guarantee can be provided on the output of a neural network. This can be very dangerous for AUVs, which can lead to damage on the platform or worse to lose in the sea. For this reason, the community is particularly interested in certifying these types of methods by using classic stability tools from control theory. Nevertheless, whenever we are using a neural network, it is not straightforward to apply the standard Lyapunov stability methodology [Lib05]. We propose here a methodology to apply such analysis to a learning-based control system similar to the ones proposed so far in this thesis. In particular, we are interested in studying how much a solution obtained from a neural network can hold some stability components without having explicitly included them in the neural network optimization scheme. The objective is to assess how different the solutions obtained from Lyapunov theory are from a Deep Learning method.

To achieve this, we consider again the RexRov2 platform described in Section 2.3.2. Because we have access to the thruster allocation matrix Eq. (2.98), the vehicle can be controlled in all of the DoFs of interest (i.e. surge, sway, heave, roll, pitch and yaw). Therefore, the control problem 3.2.3 can be framed as a double integrator:

$$u = B^{-1}[\ J^T(\mathbf{x}_v)\big(k_p e + k_i \int_0^t e(\tau)d\tau - k_d \dot{\mathbf{x}}_v\big) + g(\mathbf{x}_v) \ ], \tag{3.30}$$

where $u \in \mathbb{R}^6$ are the control inputs; the PID gain matrices $k_p, k_i, k_d \in \mathbb{R}^{6\times6}$ are the output of a neural network; $\mathbf{x}_v \in \mathbb{R}^6$ is the current UUV's state, i.e. position and orientation (Eq. 3.28); $\dot{\mathbf{x}}_v \in \mathbb{R}^6$ is the temporal derivative of the state vector; $g(\mathbf{x}_v)$ is the sum of external forces acting on the UUV's body (in our case, gravity); $e \in \mathbb{R}^6$ is the tracking error (Eq. 3.27); B^{-1} is the fixed known thrusters allocation matrix mapping the control input u into a combination of thruster power inputs resulting to the desired movement and $J^T(\eta)$ is a transformation matrix.

The principal advantage of the considered PID regulator (Eq. 3.30) is that global stability and convergence analysis of the system has been well formalized by Fossen [Fos94]. Lyapunov stability theory [Lib05] is straightforward to apply with such a control law. Following [Fos94], there exists a Lyapunov function candidate $V(x)$ for the considered UUV such that:

$$V(x) = \frac{1}{2}x^T \begin{bmatrix} M_\eta^{-1} & \alpha I & 0 \\ \alpha I & k_p & k_i \\ 0 & K_i & \alpha k_i \end{bmatrix} x, \tag{3.31}$$

where $\alpha \in \mathbb{R}$ is a small positive constant and the PID gains are $k_p, k_i, k_d \in \mathbb{R}^{6\times6}$; $M_{\mathbf{x}_v}^{-1}$ is related to the UUV's mass and can be computed from the current $\mathbf{x}_v$; we define $\mathbf{s}_t$ the control loop's state (not to be mistaken with the UUV's state): $\mathbf{s}_t = \left[p, \mathbf{x}_v, \int_0^t e(\tau)d\tau\right]^T \in \mathbb{R}^{18}$ and $p = M_{\mathbf{x}_v}\dot{\mathbf{x}}_v^T \in \mathbb{R}^6$ is the generalized momentum depending on the UUV's mass and velocity. Lyapunov stability theory [Lib05] tells us that the convergence to the steady state of the feedback loop can be guaranteed by assessing the value of the Lyapunov function (3.31) which is a function of the vehicle state. Thus, the control loop is stable at state x if and only if:

$$V(x) > 0 \text{ and } \dot{V}(x) < 0. \tag{3.32}$$

With this design, we can assess the stability of the AUV process at each time step by computing the Lyapunov function candidate Eq. (3.31).

We are also interested in the stability of the control parameters. In fact, we are often concerned by the values that a neural network can feed as output. There is little to no guarantee that these values will always remain small. Again, following Lyapunov stability theory [Lib05], there exists theoretical constraints on the gain matrices k_p, k_i, k_d and the small constant α such that local stability is guaranteed when the initial conditions of the systems are closed to $\boldsymbol{x} = 0$. In other words, if these constraints on the controller parameters are respected, we can locally ensure further stability of the feedback loop. According to the proposed Lyapunov function (Eq. 3.31), the stability of the control parameters is guaranteed (Eq. 3.32) if the following constraints are satisfied:

$$\begin{cases} k_d > M_\eta, \\ k_i > 0, \\ k_p > k_d + \frac{2}{\alpha}k_i, \\ \frac{1}{2}(1-\alpha)k_d - \alpha M_\eta + \frac{\alpha}{2}\sum_{i=1}^{6}(\eta_i - \eta_{id})\frac{\partial M_\eta}{\partial \eta_i} > 0, \\ \alpha > 0, \end{cases} \tag{3.33}$$

We found that, when limiting the parameter space to a value satisfying (Eq. 3.33), the resulting space is so small that the benefits of adaptive control are merely preserved. Therefore, we propose not to take into account the stability constraints in the parameters optimization. Our objective is then to assess to what extent the resulting solution, obtained from an ANN, can still hold some stability components. In order to facilitate the stability analysis, we want to reduce the dimension of the space depicted in Eq. (3.33). First, we can transform the constraints (3.33) into equalities as follows:

$$\begin{cases} k_d = M_\eta + M_1, \\ k_i = 0 + M_2, \\ k_p = k_d + \frac{2}{\alpha}k_i + M_3, \\ \alpha = \left\| \dfrac{-k_d}{(-k_d - 2M_\eta + \sum_{i=1}^{6}(\eta_i - \eta_{id})\dfrac{\partial M_\eta}{\partial \eta_i})} \right\|_{max} + \epsilon, \end{cases} \tag{3.34}$$

where M_1, M_2 and M_3 are three 6×6 positive matrices and ϵ is a small positive constant. With this transformation (3.34), the fulfillment of the Lyapunov stability constraints (3.33) can now be verified by only assessing the value of $[M_1, M_2, M_3, \epsilon] \in \mathbb{R}^{3\times6\times6+1}$. In order to further reduce this dimension space, we apply a diagonal transformation on the matrices M_i: $M_i = P\Lambda_i P^{-1}$, where $\Lambda_i \in \mathbb{R}^6$ are positive vectors and $P \in \mathbb{R}^{6\times6}$ is a positive invertible matrix chosen randomly beforehand. Thanks to this transformation, we can now assess the value of $[M_1, M_2, M_3, \epsilon] \in \mathbb{R}^{3\times6\times6+1}$ (and k_p, k_i, k_d with Eq.3.34) by only accessing:

$$[\Lambda_1, \Lambda_2, \Lambda_3, \epsilon] \in \mathbb{R}^{19}. \tag{3.35}$$

Design of the learning-based adaptive controller

We propose to learn a stochastic predictive model π_θ (parameterized by θ) presented by a neural network, that maps the system state vector x into the controller parameters of the PID law (as illustrated in Figure 3.4):

$$\begin{cases} \pi: \ \Omega_x \subset \mathbb{R}^{18} & \rightarrow \ \Theta \subset \mathbb{R}^{19} \\ \quad x = [\mathbf{s}_t]^T & \mapsto \ [\lambda_i, \mu_i] \end{cases} \tag{3.36}$$

The neural network takes as input the control loop's state vector $\mathbf{s}_t$ Eq. (3.31) and returns as outputs the 19 pairs (λ_i, μ_i) of mean and standard deviation that are used to model the control parameters Eq. (3.35) where each variable is represented by a Gaussian distribution $\mathcal{N}_i$ defined as:

$$\mathcal{N}_i(\lambda_i, \mu_i) = (2\pi\mu_i)^{-1/2} \exp\{-\frac{1}{2\mu_i}(x - \lambda_i)^2\}. \tag{3.37}$$

The control parameters $[\Lambda_1, \Lambda_2, \Lambda_3, \epsilon] \in \mathbb{R}^{19}$ are thus determined using the Gaussian distributions Eq. (3.37), which allow us to assess the stability of the control system. We have empirically set up an architecture composed of 2 hidden layers of 32 hidden nodes each, with the Sigmoid activation function applied to each layer. This results in a total of $(18 + 1) \times 32 + (32 + 1) \times 32 + 2 \times ((32 + 1) \times 19) = 2918$ parameters (ω) to learn from data. In a second stage of the mapping π, the PID parameters $[\Lambda_1, \Lambda_2, \Lambda_3, \epsilon]$ are obtained by sampling from the resulting Gaussian distributions $\mathcal{N}(\mu, \sigma)$. The Eq. 3.34 allows to the computation of the final PID parameters and using Eq. 3.30 the PID control inputs are derived. Accordingly, we can more easily assess the Lyapunov stability of the system. The overall control strategy is illustrated in Figure 3.4.

We propose to optimize the weights of the neural network with the Cross-Entropy Method (CEM) which is a direct search optimization approach (i.e no gradient is computed). It is an Estimation of Distribution Algorithm (EDA) inspired by Natural Evolution Strategies [Wie+08]. With CEM, during one iteration k, N sets of weights $S^k_{i=1\ldots N}$ are sampled from a Normal distribution directly in the space of weights R^{2918}. At each iteration k, N evaluations are made to determine the current best weights $S^k_{best=1\ldots N\times\rho}$ with respect to a given cost function. In our case, we used the following classic control performance (based on multi steps within an episode and connected to the tracking error (Eq.3.27) index as a cost function to minimize:

$$J = \sum_{steps} \frac{1}{6} \sum_{i=1}^{6} (\eta_{d,i} - \eta_i)^2. \tag{3.38}$$

The mean and covariance of the Normal distribution are then updated as the mean and the covariance of the $N \times \rho$ best sets of weights obtained at iteration $k - 1$. Noise σ^2_{noise} is added to the covariance of the best weights to avoid local optima. We randomly sample the next iteration weights as:

$$S^k = \mathcal{N}(\text{mean}(S^{k-1}_{best}), \text{Cov}(S^{k-1}_{best}) + \sigma^2_{noise}) \tag{3.39}$$

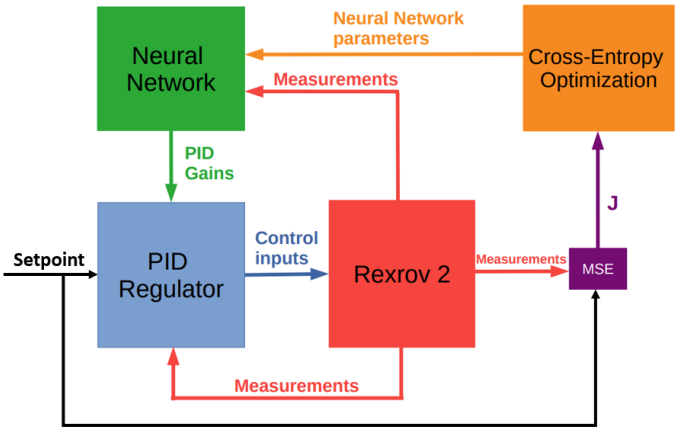
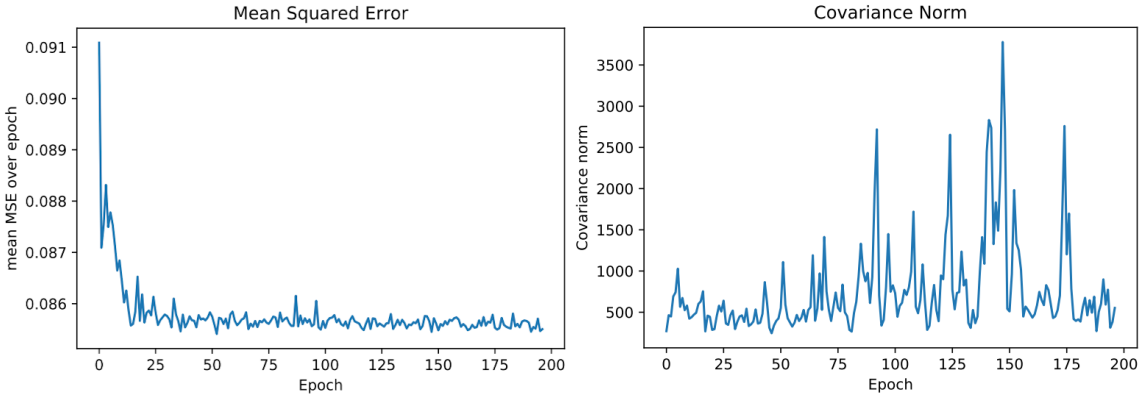


Figure 3.4: *Block diagram of the proposed method for PID tuning using Cross-Entropy Deep Learning. The control parameters are adapted during operation and are represented in the form of a function of the state of the vehicle (i.e. neural network).*

Analysis protocol

Training settings

We used the following CEM hyperparameters that have been chosen through a grid search: population size $N = 25$, proportion to keep $\rho = 0.2$ and added noise $\sigma^2_{noise} = 0.1$. Each training episode is composed of 200 time steps. One epoch is defined as performing one episode using each of the N sets of parameters candidate. The training consists in performing 200 epochs, thus a total of $200 \times 200 \times 25 = 10^6$ time steps.



(a) *Mean squared error on the setpoint.*

(b) *Covariance of the normal distribution.*

Figure 3.5: *Training curves showing the evolution of the MSE and the covariance during training.*

The training dynamics are represented with the evolution of the MSE on the setpoint in Figure 3.5a and with the evolution of the covariance of the normal distribution in Figure 3.5b. We can see in Figure 3.5a that using the CEM approach, we rapidly converge to what seems to be a minimal MSE value. The target rallying mission is successfully performed with an error on the setpoint which converges to a minimal value. On the other hand, we can see in Figure 3.5b that the covariance of the normal distribution is not converging to a minimal value. This means that the CEM algorithm has yet not converged to optimal values. A high value of this covariance means that the sphere, that is used to explore the space of parameters of the neural network, is getting bigger. This means that there remain some parameters that can be optimized. However, as we did not observe a notable gain in MSE reduction by letting the CEM optimization algorithm run longer, we proposed to stop it after 200 epochs despite this high covariance value.

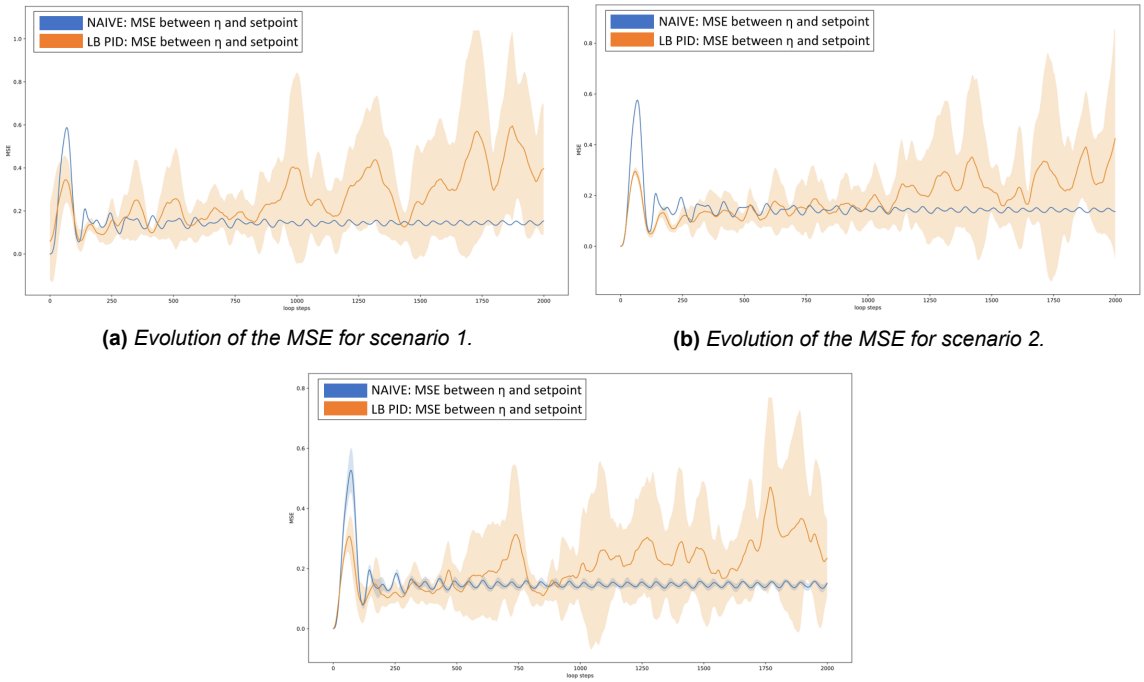
Evaluation settings

Our objective is to measure the stability ability of the AUV learning-based adaptive controller against process variation. For this objective, we propose three evaluation scenarios based on induced disturbance: none, Gaussian noise in sensor measurements, and Gaussian noise in control inputs with current disturbances denoted respectively as scenarios 1, 2, and 3. The length of the episode is now increased to 2000 time steps. The initial state of the UUV is changed at the beginning of each episode. For simplicity purposes, the proposed controller will be denoted in the following as LB PID. We compare the LB PID controller to its Lyapunov-based counterpart which consists in a PID controller whose parameters are set to $M_1, M_2, M_3 = M_i \times \boldsymbol{J}_{6\times 6}$ with:

$$M_i = \begin{bmatrix} 0.5 - 10^{-5} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 - 10^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 - 10^{-5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 - 10^{-5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 - 10^{-5} & 0 \\ 0 & 0 & 0 & 0 & 0 & (0.5 - 10^{-5}) + 10^{-5} \end{bmatrix} \tag{3.40}$$

The values used for the Lyapunov-based PID controller Eq. (3.40) have been chosen such as to satisfy as much of the constraints on the control parameters stability Eq. (3.34). The resulting controller remains adaptive (as the gains are a function of the vehicle state **x**) and will be denoted as naive PID henceforth. The only difference between these controllers is how each of these controllers adjusts the value of the parameters used to derive the PID law (3.30), making the comparison fair. The objective of the evaluation is to compare both controllers in terms of control performance but also in terms of system and parameters stability thanks to the methodology described in Section 3.2.3.

Discussion



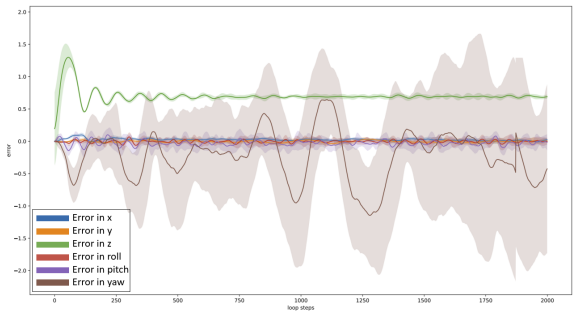
(a) *Evolution of the MSE for scenario 1.*

(b) *Evolution of the MSE for scenario 2.*

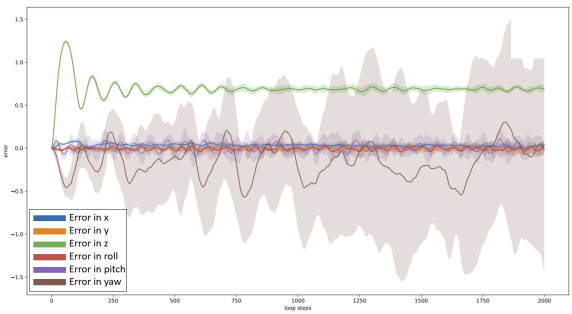
(c) *Evolution of the MSE for scenario 3.*

Figure 3.6: *Illustration of the control performance by both controllers over the different scenarios.*

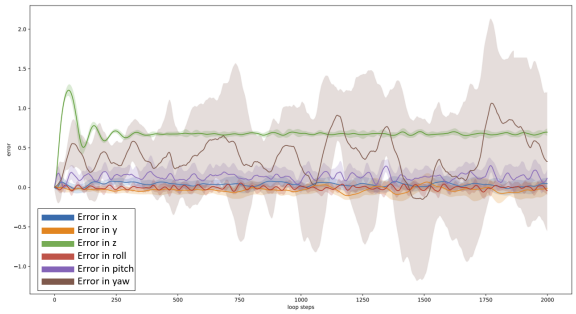
The control performance is measured as the MSE on the setpoint. The evolution of this performance is illustrated in the following figures: when facing no disturbance in figure 3.6a; when facing Gaussian noise on the vehicle position and orientation feedback in figure 3.6b and when facing Gaussian noise on the control inputs and sea current disturbance in figure 3.6c. We can see that the naive PID (in blue) results overall in better performances in terms of setpoint tracking compared to the LB PID (in orange). Nevertheless, we can see that the relative performance of the LB PID with respect to the naive PID's performance improves with increased uncertainty. In Figures 3.6b and 3.6c, we can see that the performance of the LB PID matches the naive PID for approximately 500 time steps, while without disturbance, its performance drops notably earlier as shown in figure 3.6a.



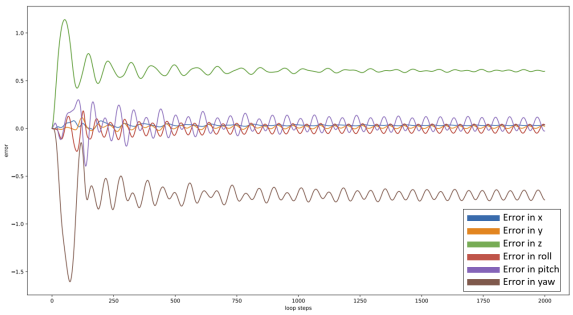
(a) *The position and orientation errors of the LB PID during an episode from scenario 1.*



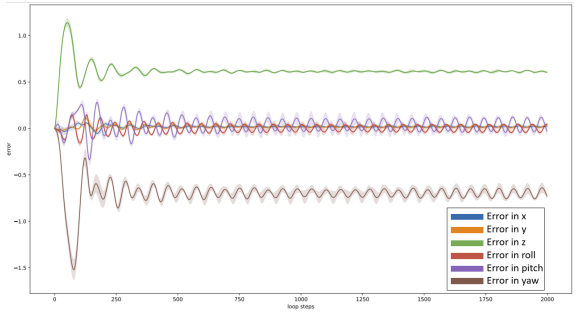
(b) *The position and orientation errors of the LB PID during an episode from scenario 2.*



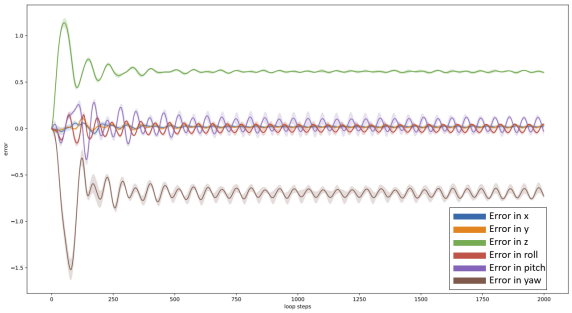
(c) *The position and orientation errors of the LB PID during an episode from scenario 3.*



(d) *The position and orientation errors of the naive PID during an episode from scenario 1.*



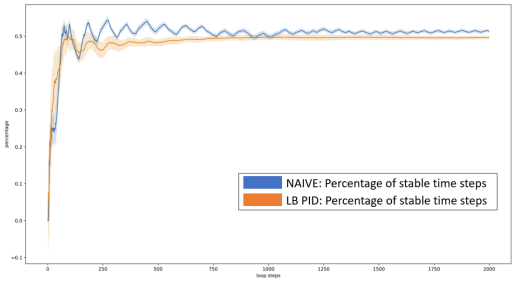
(e) *The position and orientation errors of the naive PID during an episode from scenario 2.*



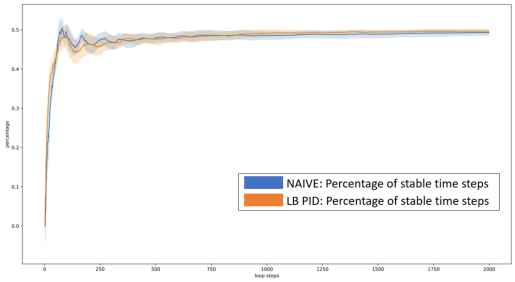
(f) *Position and orientation errors of the naive PID during an episode from scenario 3.*

Figure 3.7: *Illustration of the errors on each DoF by both controllers over the different scenarios.*

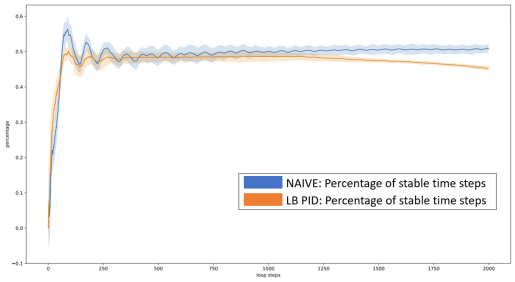
The evolution of each DoF is represented in Figures 3.7a-3.7f. We can observe a difference in regulation dynamics depending on the DoF. For instance, we can see that the depth of the UUV is not successfully regulated by none of the controllers. Due to the short latency between episodes and the position of the UUV's CoG, the UUV slightly sinks at the beginning of the episode, altering its depth and yaw angle (z, ψ). This explains the vertical drift in their associated errors observed in the figures. In addition, as seen in Figures 3.7a-3.7c, the LB PID tends to regulate effectively all DoF except the yaw angle of the vehicle ψ. This divergence of the yaw angle explains the increase in MSE observed in the previous Figures 3.6a, 3.6b, and 3.6c. We believe that the divergence of the yaw angle is due to the fact that the CEM has not yet converged to satisfying parameter values for this DoF. This could explain why the covariance of the normal distribution is still high by the end of the training. This can be due by the fact that because of the nature of the signal, the error on the yaw angle is notably smaller than the other DoFs. Because of that, the resulting MSE is dominated by the other error signals. In other words, we can still have a small MSE despite having a high error on the yaw angle. A direct improvement is to normalize in some ways the MSE so as to give an identical weight to each term in the measure of the error on the target Eq. (3.38).



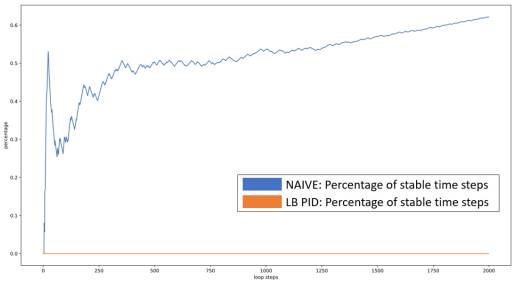
(a) *Evolution of the state stability during an episode from scenario 1.*



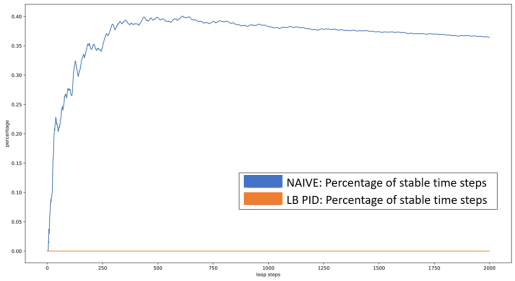
(b) *Evolution of the state stability during an episode from scenario 2.*



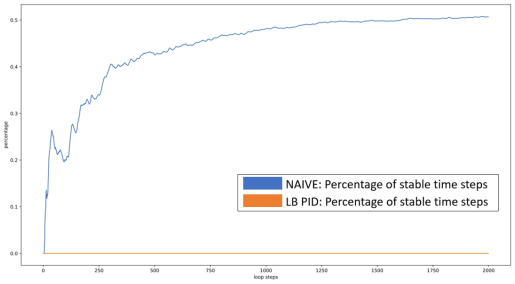
(c) *Evolution of the state stability during an episode from scenario 3.*



(d) *Evolution of the stability of the parameters during an episode from scenario 1.*



(e) *Evolution of the stability of the parameters during an episode from scenario 2.*



(f) *Evolution of the stability of the parameters during an episode from scenario 3.*

Figure 3.8: *Illustration of the state and parameters stability of each controller over the different scenarios.*

The evolution of the stability metrics is illustrated in Figures 3.8a to 3.8f. We can see in Figures 3.8a, 3.8b and 3.8c that despite not taking into account any stability constraints on the optimization procedure, the LB PID matches the system stability of the purely Lyapunov-based controller (i.e. the naive PID). This is due to the fact that the vehicle itself is fundamentally designed to be highly stable. This means that when facing no disturbance, the AUV will naturally stabilize itself to a steady state where no energy dissipation occurs. The fact that the control input is a function of a neural network has little impact on the stability of the state. Nonetheless, we can observe a divergence of the state stability in scenario 3 where we are facing sea current disturbance. This is due to the fact that it is not possible to design a vehicle that will remain stable against any possible process variations, which by definition is infinite, while the number of actuators is finite. When looking at the control parameters' stability, we observe a different trend. In fact, in the three scenarios, the control parameters of the LB PID controller are never satisfying the constraints Eq. (3.33) while the parameters of the purely Lyapunov-based controller satisfy them. With the latter, we can even notice that the control parameters' stability tends to increase over time, as the closer we are to the target, the smaller the Lyapunov-based control parameters gets and the less likely it is to diverge from it.

From this study [Koh+22], we were able to analyze and measure the stability of a learning-based adaptive controller. The main result is that despite the use of a neural network, the state stability of the AUV matches the one displayed by a Lyapunov-based controller. In the following, we will present a summary of the different suggestions rising from these preliminary studies in order to adapt the proposed learning-based adaptive control system to an AUV.

3.2.4 Suggestions

Based on the preliminary studies presented in Section 3.2, we identified three challenges that we wanted to tackle in this thesis in order to propose a satisfying learning-based adaptive control method for AUVs. These challenges guided the design of the proposed solution, and before presenting the novel learning-based adaptive control system, we list below the thought processes that led to its design:

1. Interpretability of the control system: when designing a control system, we are interested in tuning our system so as to reach the desired control performance. In control theory, these performances are traditionally expressed in terms of desired settling time, overshoot, frequency of damped oscillation, or even slew rate. These requirements are usually provided by the operators, which are the experts that will be using the AUVs, and the task of the control designer is to provide them with a control system capable of answering these needs. In the case of the PID control law, as studied in this thesis, these desired control performances can not be analyzed directly in the space of gains as they are defined in a temporal plane. For this reason, we would be interested in transforming these gain values in the space of poles which is defined in a frequency plane (using the Laplace transform). There, the aforementioned performance criterion can be easily derived and the space of pole values can be bounded so as to ensure that we meet them. Therefore, we could be able to define a region in the space of poles where the control performance is at worst equal to the desired one, and which can be interpreted physically more easily.
2. Parameters adjustment against unobservable disturbance: the principal challenge of the AUV application is that we have limited observability of the process. In the case of observable disturbances, the control parameters can be adjusted using model-based adaptive control theory such as LQR optimization [Arg+13], Gain Scheduling [Cle+02], or H2/H_∞ control [ACP06]. However, in our application, the vehicle is not equipped with sensors allowing us to measure the current disturbance. Therefore, these model-based methods can not be used. In this context, we need to use some model-free optimization methods and deep reinforcement learning. It allows us to map the pole values to a state representation of the process (which is different from the state of the AUV) that can encompass a large amount of information on the process, and thus in a model-free manner: we do not need any process model to found the optimal pole values (see Section 2.1.2).
3. Performance robustness of RL-based agent: deep reinforcement learning methods are associated to a great number of parameters that influence greatly the performance of the resulting agent. Because of their stochastic nature, these types of methods are associated with a high performance variance. A key element to control the robustness of the agent lies in the Experience Replay mechanism described in Section 2.2.6. Despite some works [ZS17][Sch+16][Fed+20] showing the effect and the limits of the original ER, it is mostly ignored by the community. Our objective is to propose a new algorithmic procedure to improve the ER technique by taking into account the characteristics of the replay mechanism taking place in biological systems. This way, the resulting methodology can be applied to any Off-Policy RL algorithm.

Based on the above suggestions, we present in the next Section 3.3 the contributions of this thesis. The objective of the theoretical and algorithmic elements is to address the aforementioned challenges in order to propose a satisfying adaptive control system for AUVs. Following the preliminary studies 3.2, we first present an augmented design of the learning-based adaptive control system that can be tuned according to to control performance requirements that can be physically interpreted. Then, we will present a novel ER technique that aims at reducing the performance variation of RL-based agents.

3.3 A novel learning-based adaptive controller

3.3.1 Design of the model-based structure

Feedback controller

We introduce now the final version of the proposed learning-based adaptive controller. This work assumes that the controlled vehicle is fully observable and controllable. This means that each of the vehicle's DoF is measurable and the desired vehicle states (within the operating regimes) are supposed to be accessible. However, the model of the process B is not available and we are facing unobserved disturbances. In this context, a PID controller is a suitable model-based structure to regulate the known part of the process. We can take into account the unknown current disturbance by considering the steady-state error variable $\sigma = \int_0^t e(\tau)d\tau$. We can rewrite the state-space equations with the augmented state vector $X = [\sigma, e, \dot{x}]$ as:

$$\frac{d}{dt}\begin{bmatrix} \sigma \\ e \\ \dot{x} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{A}} \begin{bmatrix} \sigma \\ e \\ \dot{x} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{\text{B}} u. \tag{3.41}$$

The PID state-space representation is given by:

$$\dot{X} = (A - BK)X. \tag{3.42}$$

The PID control law can then be derived as:

$$u = k_p e + k_i \sigma + k_d \dot{x}, \tag{3.43}$$

with k_p, k_i and $k_d \in \mathbb{R}^+$, anti-windup added on the integral term such as $\max(\sigma) = \max(u)$ and a low-pass filter is applied on the derivative term to reduce oscillations induced by process noise. By considering this control structure, the proposed method can be applied to any closed-loop control process where essentially we have access to feedback from the system outputs.

Adaptive direct pole-placement strategy

Among the various procedures and rules that can be applied for PID tuning [Wan05], a fundamental technique consists of assigning a set of specific values, $P = \{\lambda_1\ \lambda_2\ \dots\ \lambda_n\}$, to the eigenvalues of the feedback loop $A - BK$. Given that these eigenvalues determine the poles of all the transmittances where the associated state matrices are involved, this procedure is denoted as Pole-Placement. We can define a (normalized) control polynomial as:

$$C(s) = s^n + c_1 s^{n-1} + \cdots + c_{n-1}s + c_n, \tag{3.44}$$

whose roots are the λ_i, which can be assigned the characteristic polynomial of $A - BK$ with:

$$C(s) = det(s\mathbf{I} - (A - BK)). \tag{3.45}$$

Equations (3.42) and (3.45) yield:

$$\begin{aligned} |A - BK - \lambda\mathbf{I}| &= -\lambda(\lambda(k_d + \lambda) + k_p) - k_i, \\ &= -\lambda^3 - \lambda^2 k_d - \lambda k_p - k_i, \\ &= 0. \end{aligned} \tag{3.46}$$

To ensure minimal stability of the feedback loop (in terms of output boundness), the poles of (3.46) must be placed in the complex left half-plane. For this purpose, we only consider as eigenvalues candidates the solutions of:

$$\lambda^3 + \lambda^2 k_d + \lambda k_p + k_i = 0. \tag{3.47}$$

In order to maintain the dimension of the gain space, we propose as pole values candidates $\tau_i \in \mathbb{R}^+$ the following design:

$$\lambda_1 = \frac{-1}{\tau_1}\ ;\ \lambda_2 = \frac{-1}{\tau_2}\ ;\ \lambda_3 = \frac{-1}{\tau_3}. \tag{3.48}$$

With this formulation (3.48), we essentially propose a symmetric mapping where each pole contributes equally to the closed-loop eigenvalues. The Pole-Placement design can be written as:

$$\begin{cases} \frac{-1}{\tau_1^3} + \frac{k_d}{\tau_1^2} - \frac{k_p}{\tau_1} + k_i = 0 \\ \frac{-1}{\tau_2^3} + \frac{k_d}{\tau_2^2} - \frac{k_p}{\tau_2} + k_i = 0 \\ \frac{-1}{\tau_3^3} + \frac{k_d}{\tau_3^2} - \frac{k_p}{\tau_3} + k_i = 0 \end{cases} \tag{3.49}$$

Since τ_1, τ_2 and τ_3 are solutions to (3.47), it follows that:

$$\begin{bmatrix} 1 & \frac{-1}{\tau_1} & \frac{1}{\tau_1^2} \\ 1 & \frac{-1}{\tau_2} & \frac{1}{\tau_2^2} \\ 1 & \frac{-1}{\tau_3} & \frac{1}{\tau_3^2} \end{bmatrix} \begin{bmatrix} k_i \\ k_p \\ k_d \end{bmatrix} = \begin{bmatrix} \frac{1}{\tau_1^3} \\ \frac{1}{\tau_2^3} \\ \frac{1}{\tau_3^3} \end{bmatrix} \Leftrightarrow MK^T = N \tag{3.50}$$

The gains of the control law (3.43) are obtained by transforming back the poles with $K^T = M^{-1}N$ as:

$$k_i = \frac{1}{\tau_1\tau_2\tau_3}\ ;\ k_p = \frac{\tau_1+\tau_2+\tau_3}{\tau_1\tau_2\tau_3}\ ;\ k_d = \frac{\tau_1\tau_2+\tau_1\tau_3+\tau_2\tau_3}{\tau_1\tau_2\tau_3} \tag{3.51}$$

This mapping Eq. (3.51) is totally novel and it is the first contribution of this thesis.

Using Eq. (3.51), the bounds for the controller parameters can be defined based on control constraints that are easier to derive in the poles domain. In the present case, with the design of Eq. (3.48), for any $\tau_i > 0$, the poles of the feedback loop $A - BK$ are placed on the x-axis of the complex left half-plane, reducing the settling time and overshoot. This leaves us with the settling time requirement to define in order to set the desired distance from the y-axis and thus bound the value of the space of poles. In accordance with the considered control objective, we can define the desired maximum settling time of the closed-loop control ς as the maximum time (in seconds) after which we want the system outputs to stay around a percentage χ (e.g. $\chi = 0.05$ for 5%) of its desired values. Accordingly, the upper bound of the space of poles is derived as $\lambda_{max} = \frac{\ln(\chi)}{\varsigma}$. Therefore, as $\lambda_i \leq \lambda_{max}$ from Eq. (3.48), we can derive the upper bound of the poles:

$$\tau_{min} < \tau_i \leq \frac{1}{-\lambda_{max}}. \tag{3.52}$$

There exist a solution for all $C(s)$ (3.44) if and only if the pair (A,B) is controllable, which is assumed in this thesis. In the case of a Single-Input system, the solution is unique. In the case of Multi-Input systems, as studied here, the number of free components of the matrix K is greater than the n eigenvalue constraints. Accordingly, there exist an infinite number of solutions, among which it is not trivial to define an *Optimal* solution. In fact, depending on the process dynamics, we might favor one particular configuration of pole locations in the space (3.52) over another. The Linear Quadratic (LQ) optimization scheme then composes an alternative framework to define optimal values.

In contrast, we propose to use DRL to adapt these parameters for the control of an AUV, which consist in searching for the best values possible within (3.52) based on the process measurements. More precisely, our approach consists in using a Deep Policy Gradient method [SB18] whose objective is to explore the space of poles (3.52) in order to find at each time step the best values possible (in this space) for each control input. The policy objective is to adjust them based on the process variation and with respect to a reward function that emphasizes the control objective. This is denoted as a *Direct* method because the controller parameters are adjusted directly without the need for estimation of any process parameters. With the mapping in Eq. (3.51), the learning action space is limited to desired solutions in the poles space as illustrated in Figure 3.9. This ensures that for any pole values chosen by the ANN in the parameter cube 3.9, the resulting control input will maintain the poles of the closed loop in the left half-plane.

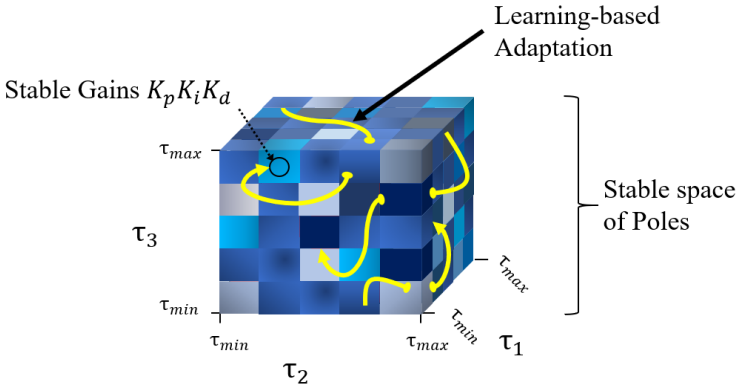


Figure 3.9: *Parameter cube representation of the proposed learning-based controller. The continuous space of possibles poles is restricted to stable locations. It is explored by a deep reinforcement learning algorithm (yellow lines) that uses these variables to compute the control input.*

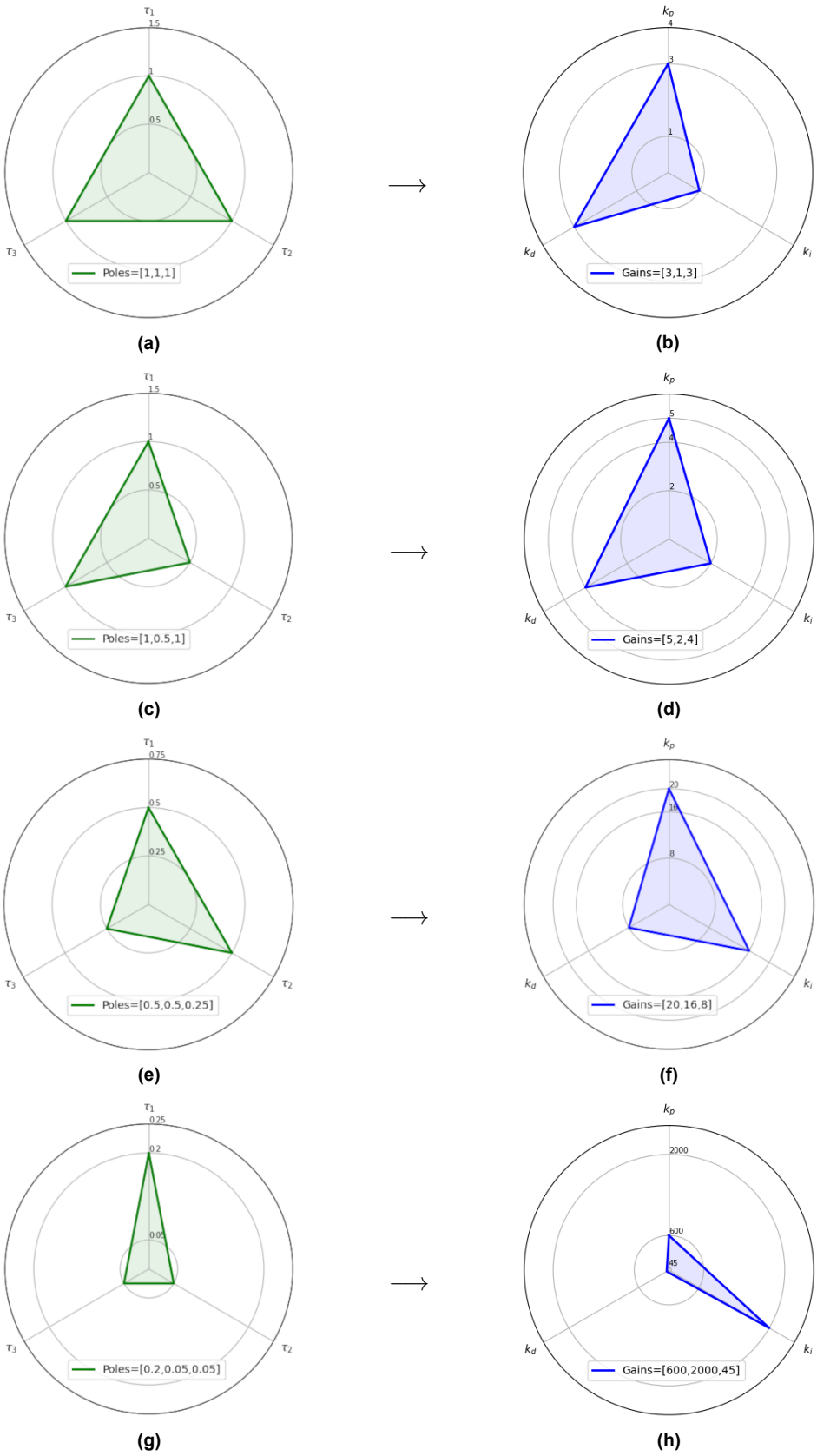


Figure 3.10: *Illustration of the resulting parameter values for a different combination of pole values. For $\tau_i \geq 1$, the resulting gains grow linearly while when $\tau_i \leq 1$, the resulting gains grow exponentially.*

Incremental VS direct pole adjustment

The continuous space of desired poles illustrated in Figure 3.9 can be explored in different ways. In this thesis we considered two approaches denoted as *Incremental* and *Direct* approaches:

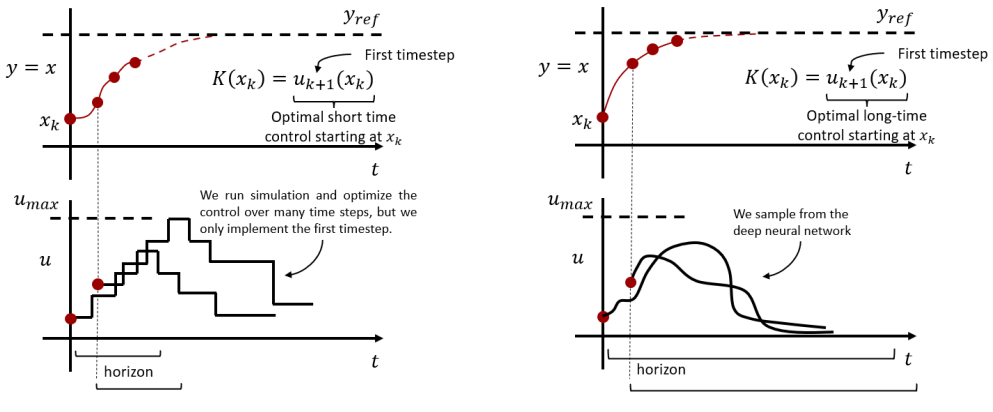
- In the incremental case, we propose to first initialize the poles at some satisfying values within the resulting space of desired solutions (which is bounded according to the desired control performance as previously explained in Section 3.3.1). The initial pole values are chosen using the Ziegler–Nichols methodology, a model-based heuristic approach. This way, we ensure that this nominal configuration provides satisfying minimal performance despite not being optimal for the complete operating regimes. Then, during operation, we adapt the pole values by adding or subtracting a small value from this nominal configuration.
- In the direct case, the poles switch from one value to another without a smooth transition. The agent can access any value of the space instantly.

The incremental procedure has the advantage of being particularly stable in terms of feedback oscillations. In fact, despite changing the controller parameters at each time step, the change being small, the feedback loop has the time to converge to a steady state.

Nevertheless, the solution obtained from the Ziegler–Nichols method is not adaptive and can only compensate for the process characteristics observed when applying the method. In particular, when the disturbances are large and/or when we have a limited measurement ability of the system state, the solution obtained from this heuristic method is not conservative enough to ensure satisfying performance over a wide spectrum of operating conditions. Therefore, with the incremental approach, it can take a notable amount of time before the control parameters move to proper values for the given operating conditions, which can often be too late. This motivated the direct pole adjustment where we can move from one pole configuration to another instantly between two time steps.

This learning-based adaptive control method can be seen as a model-free variation of model-predictive control (MPC) (see Figure 3.11). In comparison with MPC, with our method:

1. we do not need a system model, which is exactly the case study of this thesis. Because of the various disturbances and process uncertainties, we do not have a system model of our AUV process.
2. we do not rerun the entire optimization forward in time but rather infer instantly from a deep neural network. MPC relies on the fact that fast computational hardware is available to run the optimization online at every time step, which is most of the time not the case in our underwater applications. The vehicles of interest are equipped with small-size CPU cards, such as Raspberry Pi, that are not powerful enough for real-time optimization.
3. the computed action is obtained from a nonlinear function of a state representation of the overall process and not only of the system state. Therefore, the resulting solution takes into account much more information (the dimension of the process state is of the order of hundreds while the dimension of the vehicle state is of the order of tens). The resulting solution can thus more easily take into account uncertainties that are not included in the state variables of the controlled system.



(a) *MPC* **(b)** *ADPP*

Figure 3.11: *Illustration of the difference between MPC and our proposed method. The learning-based adaptive controller (b) samples the optimal actions directly from the neural network.*

3.3.2 Design of the model-free optimization algorithm

Stochastic policy

When having access to little to no information on the disturbances and when facing a time-varying process, as considered here, model-free adaptation can be exploited. In order to take into account the uncertainties in the poles selection, we propose to use DRL to build a stochastic predictive model π_θ that maps a state vector s_t into the pole values:

$$\begin{cases} \pi_\theta : \ S \subset \mathbb{R}^{dim(S)} & \rightarrow \ A \subset \mathbb{R}^{3 \times dim(u)} \\ \quad x = [\mathbf{s}_t]^T & \mapsto \ [\lambda_i, \mu_i] \, . \end{cases} \tag{3.53}$$

where $\mathcal{N}(\tau_i)$ is the probability density function of τ_i that is modeled by a Normal distribution $\mathcal{N}(\tau_i)$ as:

$$\mathcal{N}(\tau_i) = (2\pi\mu_i)^{-1/2} \exp\{-\frac{1}{2\mu_i}(x - \lambda_i)^2\}, \tag{3.54}$$

with $\lambda_i \in \mathbb{R}$ and $\mu_i \in \mathbb{R}^+$ are the mean and variance of $p(\tau_i)$ that are estimated by the Policy network. Therefore, the outputs of the Policy network are the $3 \times dim(u)$ pairs of (λ, μ) representing the Normal distributions $\mathcal{N}(\tau_i)$ used to sample the poles for each control input u_i. In practice, the current $T_i(t)$ is sampled from $\mathcal{N}(T_i)$ after applying an invertible squashing function (i.e. tanh) to $\mathcal{N}(T_i)$ (in order to bound the Gaussian distribution) and after using the change of variable to compute the likelihoods of the bounded action distribution (see Appendix C of [Haa+18c] for the complete details of this process). Designing this stochastic function (3.53) is numerically expensive due to the dimensions of the underlying spaces, excluding real-time computation with model-based methods only. The DRL framework allows us to iteratively build an estimate of this optimal mapping function.

The related methods cited later in Section 3.1 are mostly relying on the DDPG and TD3 algorithms. They are known to involve intensive tuning of hyperparameters to work properly. Additionally, in order to take into account the ocean current disturbances, one designer might favor a stochastic policy that is known to be more robust to uncertainties and to partially observable processes (at the cost of being less stable during training). For these reasons, we chose to use the Soft Actor-Critic algorithm described in Section 2.2.8. The SAC algorithm builds a stochastic policy where the action distributions are modeled by Gaussian distributions[3]

Stabilizing TD-Learning

The Mean squared error (MSE) (squared L2 norm) is the most commonly used error criterion in the context of supervised learning. However, it is not appropriate for TD-Learning defined in Section 2.2.8 where the target values (i.e. labels) move at each gradient update. Depending on the amplitude of the reward function, one can often encounter MSE in the order of tens to hundreds, and as the targets move, we are naturally more subject to sudden error increase. The associated MSE generates a high value of error derivative which can easily lead to exploding gradients and divergence. For this reason, we used the Smooth L1 loss function from Pytorch [Pas+19] for Critic's optimization as it is less sensitive to outliers compared to the standard MSE loss function. In our case, the Smooth L1 loss function acts as the L_2 loss for errors of absolute value lower than 1 (i.e. fewer oscillations when the error is low) and as the L_1 loss function otherwise (i.e. steady gradient for large error). It is less sensitive to outliers and reduces the chance of exploding gradient compare to the standard MSE loss.

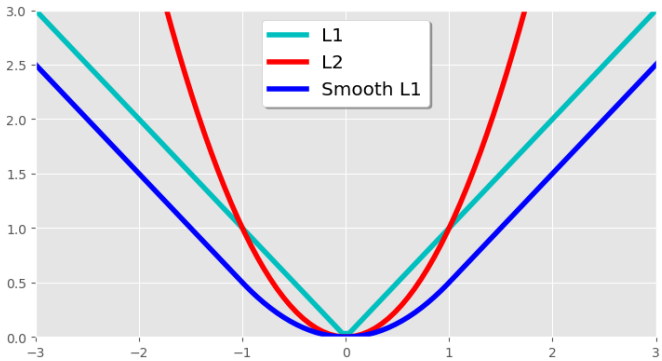


Figure 3.12: *Plots of the L1, L2 and Smooth L1 loss functions from Pytorch [Pas+19].*

[3] See appendix C of [Haa+18c] for more details on the change of variable applied to bound the Gaussian distributions.

Soft actor-critic implementation

Our implementation of the first version of the SAC algorithm is composed of 5 fully-connected Multilayer Perceptron (MLP): two Q-Value networks (with shared architectures), a State-Value and a Target State-Value networks (with shared architectures) and a Policy network. In the case of the second version of the SAC, the State-Value function is not explicitly represented by a neural network but it is approximated by the estimate (2.80) as described in Section 2.2.8. We used the same ANN architecture and hyperparameters as proposed in the original SAC paper [Haa+18c] where each network is composed of 2 hidden layers of 256 hidden units each. Therefore, no exhaustive hyperparameter tuning was required. The Pytorch framework [Pas+19] and CUDA toolkit [Nic+08] was used to implement this architecture along with an Nvidia RTX 2070 GPU card for the gradient and simulation processing. All the networks are optimized simultaneously during training, but only the policy network is used during evaluation. The neural networks are optimized using the standard Adam [KB15] method. Regularization techniques are used to prevent overfitting by reducing the variance of an ANN. It has been demonstrated that regularization does matter for Policy Gradient methods [Liu+20]. Following these results, we add regularization to the Critics only by means of weight decay of 0.001. Given the maximum entropy framework, no further regularization is recommended on the Actor [Liu+20].

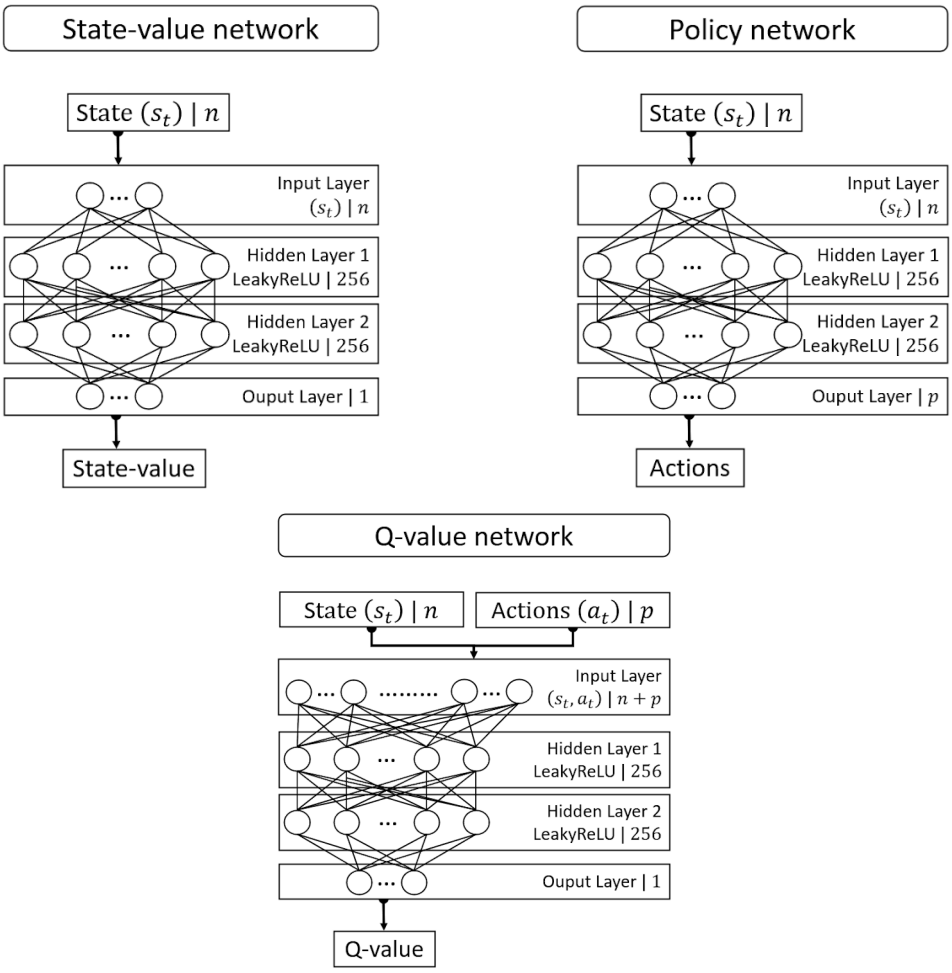


Figure 3.13: *The deep neural network structure for our implementation of the SAC algorithm. The networks are composed of dense layers only and of two hidden layers of 256 hidden nodes for each network. Each layer is a fully-connected layer represented by its type, output size, and activation function. The networks uses the same optimizer (Adam [KB15]), activation function (Leaky Relu [Xu+15]) and learning rate* $l_r = 3e^{-4}$.

As outlined in Section 2.2.9, the standard ER technique that is commonly used to optimize the above neural networks is associated with a number of challenges. In the following section, we present our contribution to the ER mechanism in order to improve its sample efficiency and stability.

3.3.3 Design of the bio-inspired experience replay

Experience Replay

Off-policy reinforcement learning tells us that we can learn the optimal policy from samples generated by any other policies. Following this heuristic, we can optimize the actor similarly to the critics, by using the agent's past experience. The state-of-the-art method to manipulate the agent's past experience in this context is called Experience Replay (ER) and was presented in Section 2.2.6.

A deeper look: the CER method

Nevertheless, even with this first ER formulation, there is a great number of parameters that are usually ignored despite having an impact on the learning performance, including:

The replay buffer size: the total number of transitions that the replay buffer can store. When its maximum size is reached, the replay buffer is accessed in a first-in-first-out fashion. The bigger the replay buffer size, the more the data will look like it is IID, which in turn improves the gradient update quality. However, if the replay buffer is too big, an important transition will have much less chance of being used to update the policy, which could impair the learning process. In contrast, if the replay buffer is too small, the learned policy can be the result of an overfitting process on recent transitions, which precludes performance improvement.

The age of a transition: the number of gradient steps taken by the agent since the transition was generated. This value can be seen as a measure of the extent to which the transitions stored in the Replay Buffer are off-policy, as it tells us how different the current policies are from those stored in the buffer. The age of the oldest policy stored increases with respect to the buffer size.

The replay ratio: the number of gradient updates per environment transition. It can be viewed as a measure of the frequency at which the agent is learning using existing data versus learning from collecting new experiences.

The size of the replay buffers, however, can impact negatively the learning performance [ZS17]. There are two competing methods that can be used to solve this issue: the Combined ER (CER) [ZS17] and the Prioritized ER (PER) [Sch+16]. CER consists of adding the latest transition performed to the mini-batch pooled over the replay buffer, whereas with PER important transitions, as measured by their associated TD-error (2.73)(2.74), are given a higher probability to be used in the gradient updates. Using CER, however, the last transition will undoubtedly be sampled and instantly affect the policy.

However, even with CER, a drop in performance was observed for certain sizes of replay buffer, at some point of the training (even when tuning the learning rate). This behavior was related to the process itself rather than to the aforementioned parameters [ZS17]. As written in [ZS17] "CER is a workaround ... and future effort should focus on developing a new principled algorithm to fully replace ER." In this paper, we propose a new ER mechanism with the ambition to decouple the performance of the agent from process complexity (as observed with CER).

Missing biological elements

A recent detailed analysis of ER was provided in [Fed+20], where an analysis of the effects of the aforementioned parameters was presented. Several conclusions on how the parameters can affect the learning dynamics were drawn, which motivated the ER design proposed in this work. These conclusions can be summarised as follows:

- Increasing the replay capacity while fixing the age of the oldest policy improves the performance because it lowers the chances of overfitting to a small subset of (state, actions).
- As the agent trains, it spends more time in higher quality regions of the environment (as measured by rewards), thus learning to better estimate the return in such regions leads to further gains in performance.
- Increasing the buffer size with a fixed replay ratio has varying improvements. The replay ratio stays constant when the buffer size is increased because of both the replay capacity and the age of the oldest policy increase. If one of these two factors is independently modulated, the replay ratio will change.

A bio-inspired experience replay

The design of the proposed approach for ER has been mostly motivated by these findings which we tried to incorporate into the ER scheme. A recent study [Hay+21] undertook a comprehensive comparison between the replay mechanism that takes place in biological brains and those in artificial learning systems. We list below some findings related to DRL only.

Replay (in biological systems) is temporally structured. Temporally correlated experience sequences are used for learning and memory combinations. This allows for more combinations of neurons which leads to faster emergence of temporal waking experiences. This feature is largely ignored by existing methods that only replay static and uncorrelated inputs.

Replay is modulated by reward and only a few selected experiences are replayed. It seems intuitive that not all experiences are useful for learning a new task. Some experiences are more important than others because they incorporate higher-quality information about the process dynamics. The challenge here is twofold: how to model this information quality and how to measure it.

Replay is treated differently for novel versus non-novel inputs. This allows for selective replay to be weighted by novelty. Biological systems tend to reduce drastically the attention given to old experiences versus that given to recent ones as the latter contains more information within them.

We propose a novel bio-inspired experience replay (BIER) technique to take into account the above-mentioned challenges while keeping in mind the requirements associated with Gradient-based optimization. In this method, the agent experience is divided into two distinct memory units (as illustrated in Figure 3.14) and samples are drawn from them differently as described next.

Sequential-Partial Memory: this buffer is denoted as B_1 and is similar to the one from the original ER scheme. Here its maximum size is set to $1,000,000$ and contains old and new transitions. We believe that, especially in the robotic case, the optimal behavior is highly temporally correlated. With robots, early actions do have an impact on future states. Let's consider for example the case of a robotic arm whose objective is to grasp a cup of coffee and place it on the other side of the table. If the robot first grasps the cup in an unsteady manner (because grasping it fast may lead to a local high reward), even if it acts perfectly afterward, there is a high probability that it will spill the contents or drop the cup along the way. Learning this temporal relationship is thus essential in order to learn what truly good behavior is, in the physical sense, is. In addition, even with very different operating conditions, the robot's behavior remains quite similar. This means that the shape of the trajectories remains within a bounded space. Therefore, our hypothesis is that learning on sequences can lead to further gains (compared to temporally uncorrelated samples) because what we learn on a trajectory can directly be applied to future ones that the agent has yet to encounter. Following this heuristic, we propose to sample random sequences of transition from Buffer B1 (i.e. transitions that are successive within the buffer since we store each of them in a *first-in-first-out* fashion). The vector of experiences E sampled from B_1 is composed of n samples as $E = [e_i\ ;\ e_{i+1}\ ;\ \dots\ ;\ e_{i+n}]$, for $0 < i < m-n$. While this replay procedure sounds very appropriate for a biological system, here we are optimizing MLPs with limited numbers of parameters and learning abilities. Using highly correlated samples often leads to repeatedly overfitting to those locally correlated samples and never really learning the true value of the functions (i.e. it ends up oscillating between different overfitting regimes). This is commonly observed in on-policy methods where the ANNs are optimized using samples generated by the same policy. For this reason, we propose to consider sequences that are only partial by storing 1 out of 2 transitions in the buffer B_1 which:

- adds a regularization effect by feeding incomplete sequences to the ANNs that encourages these networks to further learn the real value of the functions.
- reduces the age of the oldest policy contained in this buffer, which improved performance in [Fed+20].

Optimistic Memory: both bad and good behaviors are important when learning a new task because they both contain information about the process. We have been able to observe a number of cases where "positive reinforcing" is much more efficient with biological systems, for example with animal training where good and bad behaviors are respectively rewarded with treats or no response (rather than punishment). It was shown in [Fed+20] that trying to estimate values of high-quality regions (as measured by the rewards) results in better performance. In addition, as the agent learns, its performance improves and better transitions are performed. Such transitions are important because they can further improve the data collection of the agent in the future. However, as shown in [ZS17], when using a large replay buffer, such transitions are likely to influence the policy later. Their probability to be sampled decreases as the replay buffer size increases, slowing down performance improvement. Following these heuristics, our objective with this second buffer B_2 is to be optimistic about past experience, by increasing the probability of using transitions associated with such high-quality regions.

We propose to store in B_2 the upper outliers of the reward distribution that we consider to be the best transitions. Outliers can be defined according to diverse metrics depending on the nature of the variable distribution. The challenge is that we can not predict the distribution shape beforehand. For instance, with our reward function, the closer the robot gets to the setpoint, the higher the maximum value of possible reward becomes (hence, the optimal policy should lead to a reward distribution of Pearson shape). In practice, however, the closer the vehicle is to the setpoint the more difficult it becomes for it to physically reduce the errors (which is more akin to a Gaussian distribution). Depending on the system, the operating conditions, and the reward function (among others), the reward distribution can switch between various shapes, potentially making the predefined metric not robust to different distribution assumptions. Thus, we propose to consider a transition as an outlier of interest and to store it in B_2 if its associated reward $r(s_t)$ is:

$$r(s_t) > \mathbb{E}[r(s_t)], \tag{3.55}$$

where the expected value $\mathbb{E}[r(s_t)]$ is computed over the last 50,000 rewards generated that are stored as an additional variable M. The size of M was chosen in order to compute the expected reward over a moving window of approximately 100 episodes in order to give more importance to novel inputs, similar to biological systems [Hay+21]. This choice of expected value as a metric is related to the subtracted baseline in Eq. (2.77) that is the Value function, resulting in the Advantage function $A(s,a) = Q(s,a) - V(s)$. This function represents the benefits of changing the current policy as a positive value of $A(s,a)$ indicates that the evaluated pair of state-action is associated with a Q-Value higher than the expected one. Our assumption is that transitions that meet the criteria expressed in Eq. (3.55) are associated with positive values of $A(s,a)$. Using samples from the Optimistic Memory will therefore mostly improve the expected return of the policy, leading to faster discovery of successful trajectories.

The maximum size of B_2 is set to $10,000$. It is drastically smaller than $B1$ because, as the agent's performance improves over the course of training, what was considered a good transition is most likely to be outdated. Therefore, the reduced buffer size ensures that we focus on the current best transitions. Contrary to the first buffer, we propose to sample uncorrelated items from B_2 as single transitions are iteratively stored in this buffer. Finally, the mini-batch is constructed of n samples from each memory unit.

This Experience Replay mechanism illustrated in Figure 3.14 is totally novel and it is the second contribution of this thesis. In terms of algorithm complexity, as measured by the number of operations, let's consider the original ER algorithm to be associated with a complexity of $O(1)$. As presented in Appendix A, the CER algorithm is accordingly associated with a complexity of $O(2)$ and the BIER algorithm with a complexity of $O(6)$.

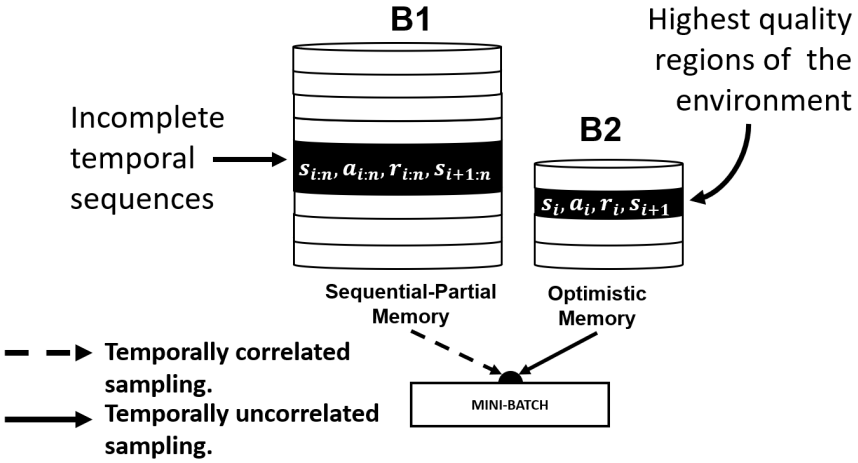


Figure 3.14: *Illustration of the proposed bio-inspired procedure. The agent's past experience is divided into two memory units: 1 out of 2 transitions are stored in the first buffer and we sample from it temporal sequences of experiences; we store in the second buffer the best transitions as measured by reward and with respect to the current policy. BIER takes advantage of the resilience from on-policy sampling while keeping the data efficiency from the off-policy formulation.*

3.4 Summary

In this section, we presented the research approach that lead to the design of the proposed learning-based adaptive control system. First, in section 3.1 we presented related works in AUVs learning-based adaptive control. We were able to identify different challenges and opportunities for contributions that we initially tried to explore with some preliminary studies presented in Section 3.2. Based on these primary results, we design a novel learning-based adaptive control system which, as illustrated in Figure 3.15, is composed of two main components:

1. First, we propose a novel adaptive pole-placement strategy where the gains of a model-based control structure are transformed in the space of poles. There, the space of possible pole values can be bounded according to desired control performance and stability requirements.
2. Secondly, the maximum entropy reinforcement algorithm called SAC (see Sections 2.2.8 and 2.2.8) is trained using the novel Bio-Inspired Experience Replay (BIER) mechanism. It consists in being more selective in the choice of past interactions to use for the update of the Critic and Actor.

As illustrated by the color code in Figure 3.15, the control system is divided into two modes: offline and online. In the online control loop (in red), the policy is exploited and the estimated poles are used to compute the PID control inputs that regulate the real platform. In the off-line learning loop (green), simulated data are used to improve the critics with off-policy TD learning using the BIER method. The policy is updated using the policy gradient of the SAC algorithm Eq. (2.77) which incorporates entropy maximization.

In the next sections, we will present how to adapt the proposed learning-based adaptive control system to AUVs. This validation was performed under simulation but also on a real platform.

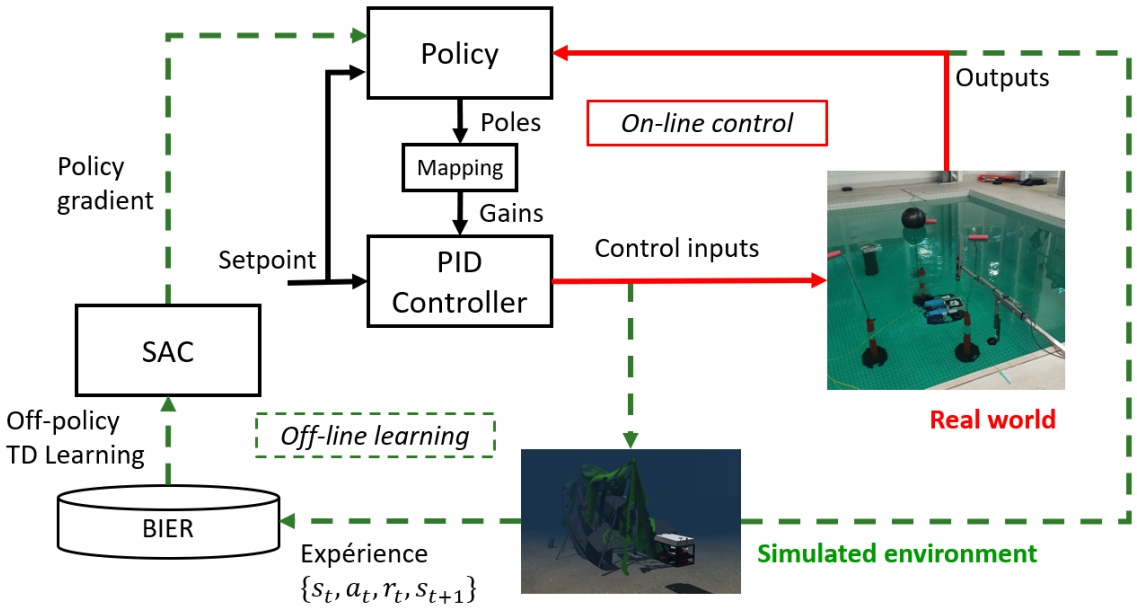


Figure 3.15: *Illustration of the overall proposed learning-based adaptive control system.*

4 Simulated validation

The final objective of this thesis is to validate the proposed method on a real platform. However, as outlined earlier throughout this thesis, DRL algorithms are known to require a big tuning effort in order to reach satisfying performance. For this reason, we perform a series of validation under simulation before applying the method to a real vehicle. We present now the simulated validation of the method proposed in Section 3.3 to AUVs using the UUV Simulator [Man+16]. In the first study, we apply solely the learning-based adaptive controller to the RexRov2 platform for a maneuvering task. Then, in a second study, we propose an improvement of the controller design for the same vehicle but for another task, along with the Bio-Inspired Experience Replay mechanism. The section ends with a summary of the findings with some suggestions for experimental validation.

4.1 Learning-based: application to AUV maneuvering under sea current disturbance

As the main focus of this thesis is AUVs, we present now how we extend the learning-based adaptive controller, that we first used on a MAV in [Cha+22] (presented in Section 3.2.2), to the RexRov2 platform. The final objective of this thesis is to use the proposed method on a real platform. To that end, we perform different validation of the method under simulation to ensure the safety and The results were published [Cha+21] and presented at the 13th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles. In the following, we summarize the main results of this paper, and its preprint version can be found in Appendix C.

4.1.1 Task description

In this study [Cha+21], we address the control problem of AUV maneuvering, which can be summarized as the stabilization of an underwater vehicle at a fixed velocity and orientation ($\mathbf{x}_{ref}$, with $\dot{x}_{ref} = 0$). The state vector is hence defined as $x = [v_x \ ; \ v_y \ ; \ v_z \ ; \ \phi \ ; \ \theta \ ; \ \psi]^T$. This is different from the previous target rallying task Eq. (3.2) because we are not regulating in terms of positions but in terms of velocities. The vehicle is fully actuated but subject to external disturbances (sea currents) that are here not observable (noted that relative current can be estimated with some types of DVL incorporating a bottom lock). Let the error between the present ($\bar{\mathbf{x}}_i$) and the desired ($\mathbf{x}_{ref_i}$) state variable be defined as $e_i = \mathbf{x}_{ref_i} - \bar{\mathbf{x}}_i$. The task of steering the AUV outputs in order to maintain the error signals within a specific threshold, over a predefined amount of time (guaranteeing the vehicle stabilization), can be achieved if the following control objective is met:

$$\forall \, t' \in [t - \varsigma, t], \ \nexists i \in \mathbb{R}^u \text{ such as } \ \mid e_i(t') \mid > \chi, \tag{4.1}$$

where $\mathbb{R}^u$ is the space of control inputs, the current time step is denoted as t and ς is the length of the period of time over which we want all the errors e_i to be less than the desired value χ. This class of control objective is used in various AUV missions, such as autonomous docking or underwater inspection, where a conservative regulation of the vehicle's outputs is required because collision can be very dangerous for both the vehicle and the environment. In this study, we consider the control task where the AUV has to maintain a constant forward velocity and orientation while being subject to sea current disturbance.

4.1.2 Design of the learning-based controller

Improved pole-placement strategy

The performance of a Pole-Placement controller is directly related to the location of its poles which makes their tuning a critical task. There exist a set of fundamental design rules to guide this choice [CG96] but in general, one should not choose closed loop poles that are highly negative, making the system fast responding (in the frequency domain) which leads to large bandwidth and thus amplification of noise. In our case, with the design (3.48), for any value of $\tau_i > 0$, the poles of the controller are placed in the left half-plane which ensures its stability. This leaves us with the settling time required to define in order to bound the value of the poles. In accordance with the control objective (4.1), we define the desired settling time of the close-loop control $\varsigma = 10$ seconds as the time after which we want the system outputs to stay within $\chi = 2.5\%$ around its desired values. These values were chosen following empirical tests when performing the Ziegler-Nichols methodology. The upper bound of the poles is then derived as

$$\lambda_{max} = \frac{\ln(\chi)}{\varsigma} = \frac{\ln(0.025)}{10} = -0.368 \tag{4.2}$$

Therefore $\lambda_i \leqslant \lambda_{max}$ (3.48), from which we can derive the upper bound of the poles

$$\tau_{min} < \tau_i \leqslant \frac{1}{-\lambda_{max}} = 2.710 = \tau_{max} \tag{4.3}$$

We choose $\tau_{min} = 0.1$ because for lower values the control inputs are too expensive in terms of control efforts and too aggressive for our control objective. Thus, the bounds of the poles are defined as

$$0.1 \leq \tau_i \leq 2.710 \tag{4.4}$$

Again, the learning objective is to estimate a policy π_μ that directly maps the pole locations from the AUV state in order to reach the control objective. The final actions are control inputs applied to control the AUV in terms of velocities and orientation:

$$\begin{cases} \pi_\theta : \ S \subset \mathbb{R}^{dim(S)} & \rightarrow \ A \subset \mathbb{R}^{3\times 6} \\ \quad x = [\mathbf{s}_t]^T & \mapsto \ [\lambda_i, \mu_i]\,, \end{cases} \tag{4.5}$$

where the dimension of the action space is 18 because we have 3 gains, thus 3 poles, for each degree of freedom. These poles candidates are then applied to compute the gains (3.51) that are used to derive the control PID law (3.43) for each control input as described in Section 3.3.2.

State vector

At each time step, the agent captures an observation vector of the process o_t as described below:

$$o_t = [\ a_{t-1}\ ;\ O\ ;\ V\ ;\ \Omega\ ;\ u_i\ ;\ e_i\ ;\ e_{L_2}\ ] \tag{4.6}$$

where

- $a_{t-1}\in\mathbb{R}^{18}$ are the estimated pole values (4.5),
- the Euler orientation of the vehicle are $O = [\phi, \theta, \psi]$,
- its linear and angular velocities are respectively $V = [v_x, v_y, v_z]$ and $\Omega = [\omega_\phi, \omega_\theta, \omega_\psi]$,
- the vector $u_t \in \mathbb{R}^6$ is composed of the current PID controller outputs,
- $e_i \in \mathbb{R}^6$ are the errors on each setpoint,
- and e_{L_2} is the Euclidean distance to the steady-state defined as Eq. (4.9).

Due to latencies and uncertainties in the process, the dynamics can become non-Makorvian which significantly degrades the learning performance. For this reason, we construct the state vector $\mathbf{s}_t$ out of the current and past 4 observation vectors (this choice of the number of frames is empirical and is motivated by related works), thus $dim(\mathbf{s}_t) = 200$.

Reward function

In accordance with the considered control objective Eq. (4.1), we design the following reward function:

$$r(s_t) = \begin{cases} r_{succeed} \textbf{ if } \forall t \in [t{-}100; t],\ \mid e_i(t) \mid \leq \chi, \\ r_{regulation}\textbf{, otherwise.} \end{cases} \tag{4.7}$$

The reward signal $r_{regulation}$ is a binary reward signal defined as:

$$r_{regulation} = \begin{cases} 20 \times e^{-(e_{L_2})^2} \times (1 + dt_{e_i(t)}) \textbf{ if } dt_{e_i(t)} > 0, \\ -20\textbf{, otherwise.} \end{cases} \tag{4.8}$$

where e_{L_2} is the Euclidean distance to the steady-state:

$$e_{L_2} = \sqrt{\sum_{i=1}^{i=dim(u)} e_i^2(t)} \tag{4.9}$$

and $dt_{e_i(t)}$ is the derivative of the error e_i computed over two time steps such as:

$$dt_{e_i(t)} = e_i^2(t-1) - e_i^2(t) \tag{4.10}$$

This reward function highly recompenses error decrease and penalizes any deviation from the Euclidean path to the steady state. If the control objective is met, the reward $r_{succeed} = 500$ is generated which ends the current episode.

4.1.3 Training

The training consisted in performing a total of 1 million time step iterations. Each episode has a maximum length of 300 time steps (equivalent to 20 seconds). This parameter also required tuning because depending on the problem, an overflow of steps can highly degrade and slow the learning, while too few steps often cause the policy to over-fit. The complete list of training hyperparameters is provided in Table 4.1 with the details on the DRL framework. A training episode is defined as follows:

1. At the beginning of the episode the AUV is initialized at the position $(x_0, y_0, z_0) = (0, 0, -40)$ with null velocity and a random orientation $(\psi_0, \theta_0, \phi_0) \in [\frac{-\pi}{4}; \frac{\pi}{4}]$.
2. A random set of current variables is generated such as $v_c \in [0, 0.5]$ and $[h_c, j_c] \in [\frac{-\pi}{4}; \frac{\pi}{4}]$ which are then kept constant during the episode.
3. A random vector of setpoints is generated such that $\Lambda_{ref} = [v_x, 0, 0, 0, 0, 0]^T$ with $v_x \in [0.5, 1.5]$.
4. Then, the off-policy exploration strategy is used and the episode ends when the step number reaches 300 or $r_{succeed}$ is generated.

Table 4.1: *List of hyperparameters and their values.*

Training hyperparameter	Value
SAC version	1 (see Section 2.2.8)
Activation function	Leaky ReLU
Optimizer (all networks)	Adam [KB15]
Learning rate (all networks)	3×10^{-4}
Discount factor (γ)	0.99
Mini-batch size	256
Target network smoothing coefficient (Δ)	0.005 (see Section 2.2.8)
Update frequency (all networks)	1
Critics L2 regularization	0.001
Layer Normalization [BKH16] (all networks)	True
Reward scale	40
Automatic temperature adjustment	False
Replay buffer max size	$1e6$
Replay start size	$1e4$
Experience Replay method	CER (see Section 3.3.3)

4.1.4 Evaluation

For comparison, we used the optimal PID controller provided in the UUV Simulator to show the benefits of our approach. It is a 6-DoF PID controller whose parameters have been optimized using SMAC [HHL11]. Both controllers are therefore based on the exact same structure (PID type control law) for a fair comparison. Evaluation outcomes are provided in Tables 4.2, 4.3, and 4.4. We use the following metrics: the success rate, the mean reward per step and the positive reward rate (as described in Section 3.2.2) computed over all episodes for each scenario. We defined three evaluation scenarios as follows:

- Scenario 1: the setpoints are fixed during the whole episode while the sea current variables vary.
- Scenario 2: the sea current variables are fixed during the whole episode while the setpoints vary.
- Scenario 3: both sea current variables and setpoints vary during the episode.

When varying, these variables have a sinusoidal shape defined as:

$$[v_x, h_c, j_c, v_c] = D_1 \times sin(D_2 \times t) + D_3, \tag{4.11}$$

with $D_1 \in [0.1; 0.25]$, $D_2 \in [0.5; 1]$ and $D_3 \in]1; 1.25]$ for v_x and $D_1 \in [0.25; 1]$, $D_2 \in [0.5; 1]$ and $D_3 \in]0; 0.5]$ for $[h_c, j_c, v_c]$ randomly chosen. The evaluation consists in performing 500 episodes for each scenario with a maximum step size per episode of 500 and different from each other in terms of the above-mentioned variables. The outcomes of this evaluation are provided in Tables 4.2, 4.3, and 4.4 with the success rate, the mean reward per step and the positive reward rate.

4.1.5 Discussion

In terms of success rate, the proposed learning-based adaptive controller is exhibiting the best performance. Compare to the OPF controller, the success rate of our learning-based adaptive controller is ~ 4 times higher

Table 4.2: *Success rate.*

Scenario	OFP	DR-DAPP
1	22%	**88%**
2	7%	**79%**
3	5%	**56%**

Table 4.3: *Mean reward per step.*

Scenario	OFP	DR-DAPP
1	9.398	**15.139**
2	-1.226	**12.110**
3	0.878	**6.200**

Table 4.4: *Positive reward rate.*

Scenario	OFP	DR-DAPP
1	75.3%	**91.7%**
2	50.3%	**87.2%**
3	54.0%	**81.0%**

in the first scenario, and ~ 11 times higher on the second and third scenarios. When the disturbance varies over time, the OPF controller is not able at all to compensate for it with its fixed parameters. The learning-based controller on the other hand is able to adapt to variations in the current disturbance. We can see that the OPF controller is not able to complete the task despite its ability at keeping the error very low. The numerous variation in the process makes the task of maintaining the error within a threshold challenging for a controller with fixed parameters. In fact, if only one DoF exceeds the threshold value, the counter is reset to zero, and to objective is not completed.

In terms of reward per step, the learning-based controller is outperforming the OPF controller with a value that is 1.61, 10.87, and 7.06 times higher than the OPF controller (respectively for scenarios 1, 2, and 3). The positive reward rate displayed by the learning-based controller is also the highest in every evaluation scenario. It is 1.21, 1.73, and 1.5 times higher than the OPF controller (respectively for scenarios 1, 2, and 3). When comparing the scenarios between each other, we can see that the performance difference is low between scenarios 1 and 2 and is large when comparing scenario 3 with the other scenarios.

As we have been able to observe, the drop in performance is larger in scenario 3 where there is process and disturbance variation. In order to identify which process variations have a big impact on the agent performance, we propose next to evaluate the agent on more various scenarios. We present in the following section improvements on the design of the model-based and model-free parts as well as on the reward function and the exploration ability of the agent. We will also evaluate the proposed BIER method against the standard CER method.

4.2 CER vs BIER: application to AUV regularization under sea current disturbance

In the previous study [Cha+21], we were able to observe that the agent performance varies a lot against increasing process variation. In order to identify which process variation the performance is most sensitive to, we propose in the next study to improve the AUV learning-based adaptive control system and to evaluate it to a greater variety of scenarios. In addition to an improved controller design, we also evaluate the proposed BIER method against the standard CER technique. The first objective of this study is to identify which process variation causes the drop in performance in order to design a proper incremental environment complexity [Cha+20a] technique which will allow an easier sim-to-real transfer of the policy. Secondly, the objective is to evaluate the BIER method in terms of training dynamics and evaluation performance. The results were summarized in the paper [CHA+22] whose preprint version can be found in appendix C.

4.2.1 Task description

In this study [CHA+22], we addressed the exact same AUV maneuvering task described earlier in Section 4.1.1 that we previously studied in [Cha+21].

4.2.2 Improvement of the learning-based controller

In accordance with the control objective shown in Eq. (4.1), and due to the lower performance observed against highly varying disturbances in [Cha+21], we define the desired maximum settling time of the closed-loop control $\varsigma = 10$ seconds as the maximum time after which we want the system outputs to stay around $\chi = 5\%$ (and not 2.5% as we did in [Cha+21] because it was too small) of its desired values. The upper bound of the space of poles is derived as:

$$\lambda_{max} = \frac{\ln(\chi)}{\varsigma} = \frac{\ln(0.05)}{10} = -0.2995 \tag{4.12}$$

Therefore, as $\lambda_i \leq \lambda_{max}$ from Eq. (3.48), we can derive the upper bound of the poles:

$$\tau_{min} < \tau_i \leq \frac{1}{-\lambda_{max}} = 3.338 = \tau_{max} \tag{4.13}$$

Again, we set $\tau_{min} = 0.025$ because, for lower values, the control inputs are too expensive in terms of control efforts and too aggressive for our control objective. Thus, the bounds of the poles are defined as:

$$0.025 \leq \tau_i \leq 3.338 \tag{4.14}$$

State vector

At each timestep, the agent captures an observation vector o_t representing the process dynamics that we defined as:

$$o_t = [a_{t-1}\ ;\ \Theta\ ;\ V\ ;\ \Omega\ ;\ u_t\ ;\ e_t\ ;\ e_{L2}\ ;\ d_{rate}\ ;\ \delta_\chi], \tag{4.15}$$

where

- $a_{t-1} \in \mathbb{R}^{18}$ are the last action estimated (i.e. poles value),
- $\Theta = [\phi; \theta; \psi]$ are the Euler orientation of the vehicle (roll, pitch, and yaw respectively),
- $V = [v_x; v_y; v_z]$ and $\Omega = [\omega_\phi; \omega_\theta; \omega_\psi]$ are its linear and angular velocities,
- $u_t \in \mathbb{R}^6$ are the last control inputs applied,
- $e_t \in \mathbb{R}^6$ are the error values on each setpoint,
- e_{L2} and $dt_{e_i(t)}$ as described in Section 4.1.2,
- and $\delta_\chi \in [0, 1]$ is a variable that keeps track of the number of successive steps where all the errors are within the threshold (i.e. if $\delta_\chi = 1$, the control objective is achieved).

The dimension of the observation vector o_t is therefore equal to 42. Noted that with this observation vector (4.15), the current disturbance characteristics are not included. In order to improve the process observability and follow our previous results [Cha+20a], we construct our state vector s_t out of the current and past observation vectors along with their two-by-two difference. This results in a 126-dimensional state space defined as:

$$s_t = [o_t\ ;\ o_{t-1}\ ;\ o_{t-1} - o_t]. \tag{4.16}$$

Reward function

The following terminal reward signal is defined to take into account the long-term stabilization requirement:

$$r_{success} = 1000 \ \textbf{if} \ \forall\, t \in [t{-}100, t], \ | \, e_i(t) \, | \leq \chi. \tag{4.17}$$

Therefore, performing the desired control objective, depicted in Eq. (4.1), will generate this reward, Eq. (4.17), and its value was chosen in order to make sure that, for all trajectory lengths, the maximum sum of return is obtained only by stabilizing the vehicle. Otherwise, the reward $r(s_t)$ is generated. Let's define the Euclidean norm of the errors vector as $e_{L2}(t) = \sqrt{\sum_{i=1}^{i=dim(u)} e_i^2(t)}$ and its derivative is computed over the last two frames and denoted as $d_{rate}(t)$. The reward $r(s_t)$ is then defined as:

$$r(s_t) = C_1 \times \exp\left[-(e_{L2}(t) \times C_2)^2\right] \tag{4.18}$$

Here, we empirically chose the reward scale as $C_1 = 40$, which gave us the best performance, by following advice from [Haa+18c]. The reward signal, Eq. (4.18), is equal to its maximum value possible per step (that is C_1) only when all the current errors are equal to zero. As AUVs move slowly, successive states display error signals $e_i(t)$ of minor and similar amplitude. We find that this addition of $C_2 = 10$ (compared to the previous work [Cha+21]), makes it easier for the critics to differentiate the State-Value of successive states without altering the reward scale as $\lim_{x \to 0} C \times e^{-x} = C$. This reward function, Eq. (4.18), encourages the agent to reduce the errors as much and as fast as possible and the vehicle stabilization is further promoted by generating the maximum reward possible per step.

Exploration strategy

We used the adaptive parameter noise [Pla+18] technique (as described in Section 2.2.5) with the following values: the metric $\mathbb{E}_s[\cdot]$ is estimated over a batch of 1000 samples from the Replay Buffer, the initial standard deviation is 0.60, the threshold is set to 0.10 and the update rate is set to $\alpha = 1.01$.

4.2.3 Training

The training consists in performing a total of 3000 episodes. Each episode has a maximum length of 500 time steps (equivalent to ∼35 seconds). The training episode characteristics are defined exactly as described in Section 4.1.3. We compare two versions of the learning-based adaptive controller, one that uses the CER method and one that uses the BIER method. Again, we compare the resulting policies with the OFP controller from the UUV Simulator defined in 3.2.1. The complete list of training hyperparameters is provided in Table 4.5 with the details on the DRL framework.

Table 4.5: *List of hyperparameters and their values.*

Training hyperparameter	Value
SAC version	1 (see Section 2.2.8)
Activation function	Leaky ReLU
Optimizer (all networks)	Adam [KB15]
Learning rate (all networks)	3×10^{-4}
Discount factor (γ)	0.99
Mini-batch size	256
Target network smoothing coefficient (Δ)	0.005 (see Section 2.2.8)
Update frequency (all networks)	1
Critics L2 regularization	0.001
Layer Normalization [BKH16] (all networks)	True
Reward scale	40
Automatic temperature adjustment	False
Replay buffer max size	$1e6$
Replay start size	$1e4$
Experience Replay method	CER/BIER (see Section 3.3.3)

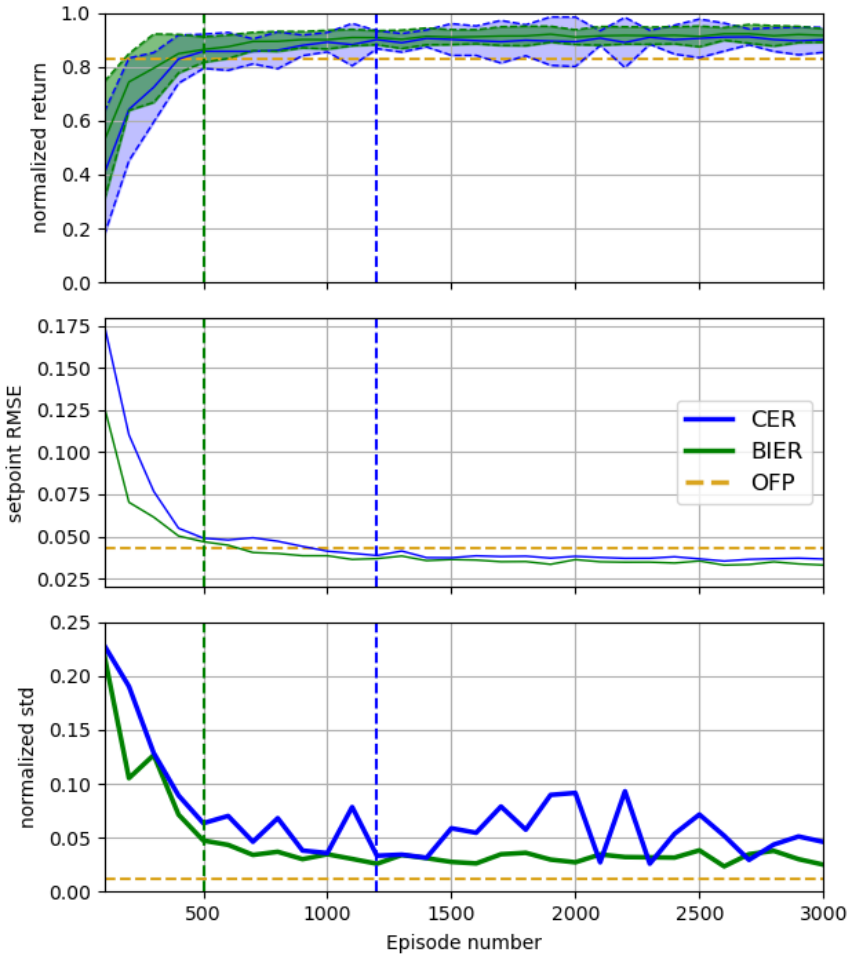


Figure 4.1: *Training curves for both Experience Replay methods. The BIER agent outperforms the CER agent in both learning speed and variance despite having access to a reduced variety of samples. Both learning-based controllers outperform the OFP controller).*

The training curves are provided in Figure 4.1 with first the normalized return, the setpoint RMSE, and the normalized standard deviation of the return. Learning-based adaptive controllers are represented in blue and green with respectively the CER and BIER methods used for the off-policy TD learning. The model-based counterpart of the control structure, namely the OFP controller, is represented in yellow with a dashed line. The performance of the OFP controller is the mean value obtained over 500 random training episodes. As we can see in the first plot of Figure 4.1, both methods are able to learn the task and converge toward what seems to be a maximum value of the reward. In the second plot of Figure 4.1, the control performance is displayed in terms of RMSE on the setpoint. We can see that both learning-based adaptive controllers exceed the control performance of the model-based controller, which is represented by the vertical lines. The agent trained using the BIER method was able to exceed the performance of the OFP controller, in less than half the number of episodes compared to the standard CER method. In the third plot of Figure 4.1, the evolution of the return standard deviation is represented. We can see that the performance improvement is smoother with the BIER method which exhibits a lower reward standard deviation (which tends to reduce over time contrary to the CER agent). With the CER method, the variance is higher, with spikes that even drive the agent to lower performance than those obtained with the OFP controller. Noted that the OFP controller displays the lowest standard deviation (third plot of Figure 4.1), thanks to the model information incorporated in the SMAC [HHL11] method. The results show that despite only manipulating the past experience differently, we can, with the BIER method, learn faster and with improved stability (i.e. lower standard deviation).

4.2.4 Evaluation

In order to visualize the effect of process variation, we present below different evaluation scenarios of increasing complexity as measured by desired setpoint and current velocity.

Scenario 1: the setpoint range is the same as during training but no current disturbances are applied. This scenario is therefore in theory simpler than the training one.

Scenario 2: the process is the same as during training but with setpoint and current variables that were not seen during training (but are still in the same range).

Scenario 3: the setpoint v_x is increased to the range [0.5,1.0], and the current variables are the same as during training.

Scenario 4: the setpoint characteristic is the same as during training but the current velocity v_c is increased to [0.5,1.0] $(m.s^{-1})$ and $(h_c, j_c) \in [-\pi, \frac{-\pi}{2}[\cup]\frac{\pi}{2}, \pi]$.

Scenario 5: both the setpoint and current variables are increased to the range defined in scenarios 3 and 4.

Scenario 6: we keep the increased range from scenario 5, and at a random timestep during the episodes (between the 100th and 400th timestep), we vary the current variables (velocity and orientation) within the same training range.

In Table 4.6 and 4.7, the metrics for each scenario were computed over 500 episodes (different from each other). The line “Baseline” denotes the performance obtained at the end of the training, which incorporates the exploration strategies described in Section 4.2.2. For the other scenarios, the exploration is removed. We propose in Figure 4.2 a bar chart of these results to simplify the analysis of the results.

Table 4.6: *Mean RMSE per step.*

Scenario	CER agent	BIER agent
Baseline	0.0364	**0.0330**
1	0.0370	**0.0366**
2	0.0350	**0.0320**
3	0.0448	**0.0418**
4	0.1483	**0.1214**
5	0.1656	**0.1508**
6	0.1802	**0.1637**

Table 4.7: *Normalized mean return.*

Scenario	CER agent	BIER agent
Baseline	0.9104 $\pm$ 0.0461	**0.9219 $\pm$ 0.0250**
1	0.9072 $\pm$ 0.0309	**0.9347 $\pm$ 0.0262**
2	0.9108 $\pm$ 0.0456	**0.9244 $\pm$ 0.0240**
3	0.8774 $\pm$ 0.0416	**0.9124 $\pm$ 0.0266**
4	0.4078 $\pm$ 0.2965	**0.5071 $\pm$ 0.2530**
5	0.3556 $\pm$ 0.2846	**0.4289 $\pm$ 0.2573**
6	0.3238 $\pm$ 0.2219	**0.3966 $\pm$ 0.2167**

4.2.5 Discussion

The CER and BIER agents are able to stabilize the vehicle over the first 3 scenarios where the performance is matching the training one. When increasing the desired setpoint value (i.e. scenario 3), we can see the associated RMSE slightly increasing but the performance remains satisfying. The performance drop is the largest when increasing the sea current disturbance (i.e. scenarios 4, 5, and 6). The sensibility to this disturbance is further depicted by the performance difference that is much smaller between scenarios 5 and 6 compare to between scenarios 3 and 4. We believe this is due to the current characteristics not being explicitly included in the state vector. Therefore, since the agent did not experience such disturbance during training, its performance drop drastically in these scenarios (i.e. scenarios 4, 5, and 6).

The BIER agent performs better in scenario 1 compared to scenario 2 which is simpler than the training scenario as it does not incorporate sea current disturbance. This suggests that overfitting is not happening, otherwise even in a simpler scenario, the performance would be lower than the training one (i.e. scenario 2). More interestingly, its performance compared to CER has also increased in scenarios 5 and 6, despite the distribution shift being particularly large there. This suggests that the policy obtained with the BIER method has better generalization abilities as its performance increased even on scenarios very different from the training one. We can see that the OFP controller is much more sensitive to sea current disturbance compare to setpoint variation (as the latter is included in the model). This tendency was already observed in the previous paper [Cha+21] (presented in Section 4.1.4) but we can clearly see here that disturbance variation can be compensated more with the learning-based adaptive controller. The learning-based controllers are able to exceed the OFP controller in scenario 6, despite not having experienced such disturbances during training.

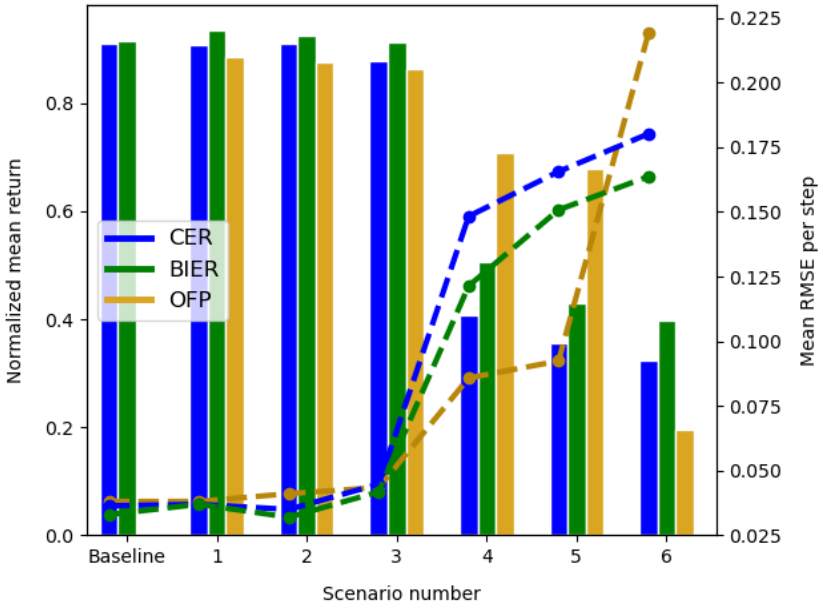
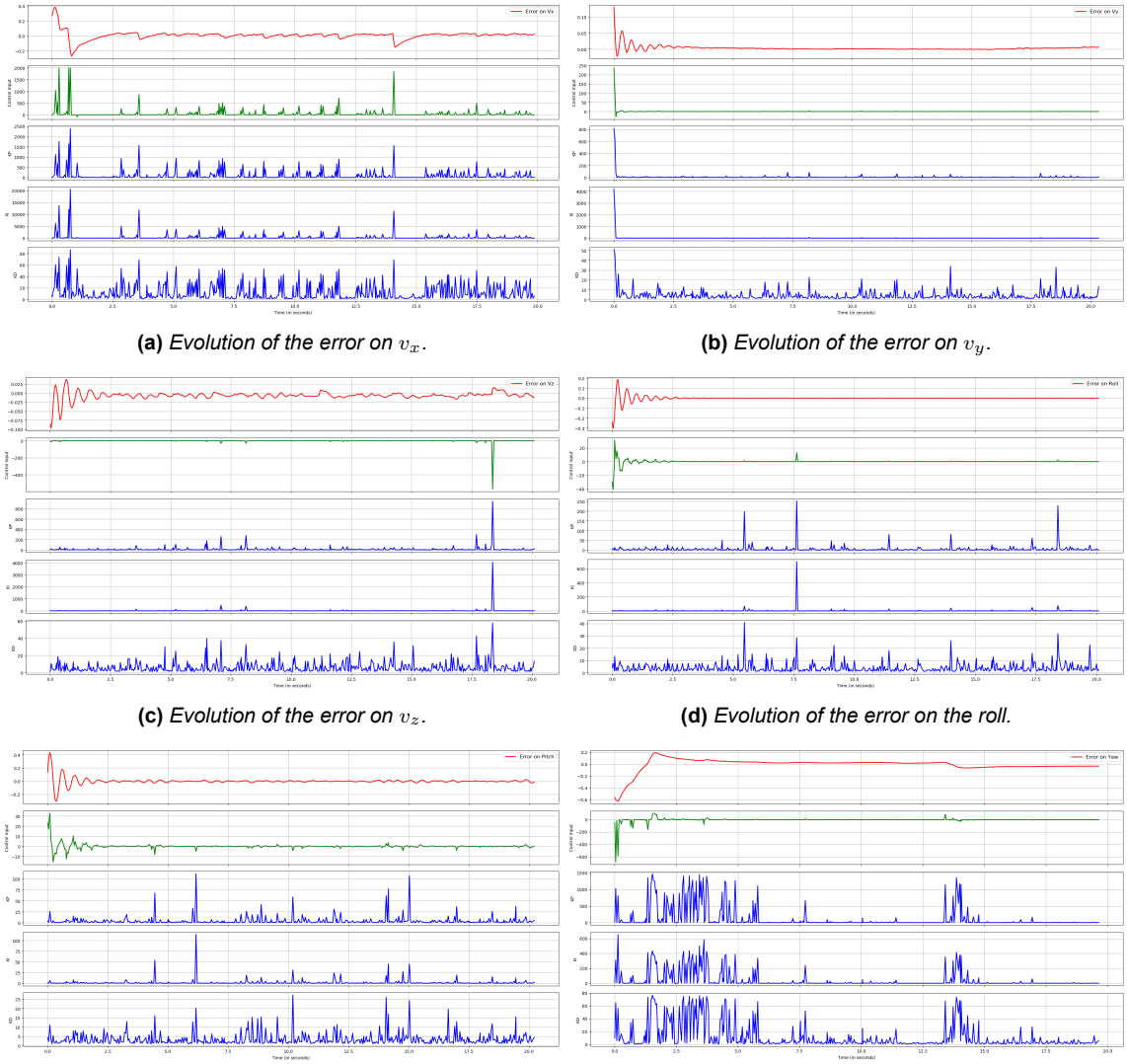


Figure 4.2: *Illustration of the evaluation performance. The only difference between CER and BIER is how, during training, the agent's past experience was used. Noted that, despite having been trained only under process variation from scenario 2, the learning-based adaptive controllers outperform the nonadaptive optimal model-based controller on scenario 6 where the distribution shift is particularly large.*



(a) *Evolution of the error on* v_x.

(b) *Evolution of the error on* v_y.

(c) *Evolution of the error on* v_z.

(d) *Evolution of the error on the roll.*

(e) *Evolution of the error on the pitch.*

(f) *Evolution of the error on yaw.*

Figure 4.3: *Evolution of the error on each DoF for an episode from scenario 5.*

In Figures 4.3a to 4.3f we plot the evolution of the error on each DoF for an episode from scenario 5. In this scenario, despite not having seen such values of the desired setpoint and current velocity, the learning-based adaptive controller is still able to compensate for these uncertainties and to maintain the errors close to 0. We can see that the controller parameters vary differently depending on the DoF which shows that some DoF is more sensitive to current disturbance.

4.3 Summary

In this section, we presented the simulated validation of the proposed learning-based adaptive controller. First, we described how we adapt the control design for an AUV with an improved pole-placement strategy. The resulting control system was then evaluated under simulation for an AUV maneuvering task. From the results of this first validation, we are able to see that the proposed method is doing better at stabilizing the AUV compared to a purely model-based optimal, but nonadaptive, version of the exact same control structure. We noticed that the performance of the learning-based adaptive controller drastically drops with increased process variation and proposed to study the cause of this incident.

Then, we proposed a second validation study to evaluate the performance of the proposed control system against more process variation. We also evaluated at that time the proposed experience replay method by training the exact same control system with the standard CER method or with the proposed BIER strategy. From these simulated results, we are able to see that the BIER method leads to faster training and more stable performance stability during training and evaluation. We are also able to identify that it is the change in setpoint and current disturbance that is hard to compensate for the learning-based adaptive control system.

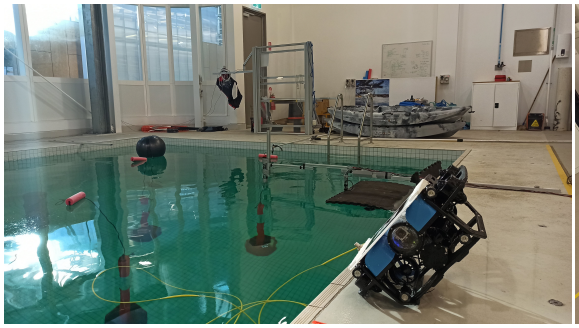
After this simulated validation, the next step is to validate the proposed method on a real platform. In the next section, we will present our protocol for real-life validation with a presentation of an improved design of the control system and the experimental protocol.

5 Experimental validation

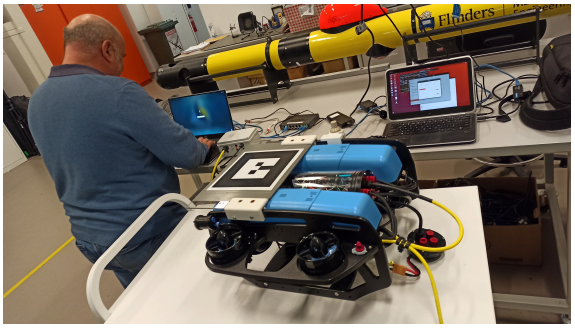
A major challenge in reinforcement learning is to transfer successfully the policy to the environment of interest. There has been a lot of effort in the development of DRL as it becomes more and more efficient to train. The bottleneck of DRL remains the sim-to-real transfer. As outlined in [Dul+20], real-world reinforcement learning is associated to a number of challenges:

- Learning on the real platform limits the number of samples and is often not feasible. The real systems of interest are often slow-moving, fragile, or expensive enough to operate such that the data they generate is too costly to consider current methods.
- Most real robotic systems have delays in either sensing, actuation, or reward feedback. These can be caused by some safety checks or just because depending on the system, the effect of actions can take a long time to manifest (which is particularly the case with AUVs).
- Many practical real-world processes have large and continuous state and action spaces which can represent critical issues for RL algorithms [Dul+15][Tes+19].
- Partial observability and non-stationary that in the case of AUVs appear as noise (e.g. uncalibrated/broken sensors) or as stochasticity (e.g. the vehicle behaves differently at each session).

In this section, we present the results of our campaign of experimental validation illustrated in Figure 5.1, where we transfer our policy on a real robot, emphasizing approximately 280 minutes (or $\sim$4h40) of operating time, which took us several months to complete and where we encounter most of the above challenges.



(a) *The BlueRov2 platform resting on the side of the water tank at FLINDERS University.*



(b) *A Kalman filter fuses IMU and camera data to estimate the marker's position.*

(c) *Jonathan Wheare and myself monitoring the robot from the computer station.*

(d) *Andrew Lammas adjusting the disturbance system before an experimentation session.*

Figure 5.1: *The experimental validation took several months and required the efforts of a team. Helped by Jonathan Wheare, Andrew Lammas, and by all my supervisors, I was able to perform real-life validation of the proposed learning-based adaptive controller showing the gains in performance compared to the standard method and its ease of deployment.*

5.1 Task description

In accordance with the sensors available on the vehicle and the pool characteristics, we decided to address the control problem of multi-stations keeping that is to stabilize the vehicle, for a given amount of time, successively at different setpoints in space that is represented by a position and orientation. The state of the vehicle at the timestep t denoted as s_t is defined by its Cartesian position and Euler orientation $s_t = [x_t\ y_t\ z_t\ \psi_t\ \theta_t\ \phi_t]^T$ (respectively roll, pitch and yaw for its orientation). A setpoint is defined as $s_w = [x_w\ y_w\ z_w\ \phi_w]^T$. The control objective is to minimize the Euclidean distance d_t between the AUV and the setpoint:

$$d_t = \sqrt{(x_w - x_t)^2 + (y_w - y_t)^2 + (z_w - z_t)^2 + (\phi_w - \phi_t)}. \tag{5.1}$$

The task of station keeping can be achieved if the following control objective is met:

$$\forall\ t' \in [t - \varsigma, t],\ \nexists i \in \mathbb{R}^u \text{ such as } \ \mid e_i(t') \mid > \chi, \tag{5.2}$$

where $d_{reached}$ is the desired threshold value that we want the Euclidean distance d_t to be lower than for the setpoint to be considered reached. This class of control objective is used in various AUV missions, such as autonomous docking or underwater inspection. In our case, as described in the following, to test the performance of our control system, we defined multiple setpoints where we want the vehicle to stabilize. The experimental validation consists in performing station keeping at several setpoints in space. Starting from an initial position, the vehicle is required to perform station keep for an amount of 1000 timesteps at each setpoint (which is equivalent to 45 seconds). One session is, therefore, equivalent to ~ 7 minutes and consists in performing the station-keeping task illustrated in Figure 5.2. We evaluated our learning-based adaptive control against two environment configurations: with and without current disturbance. In order to provide significant value to our results, we performed this station-keeping task 10 times for each control system and for each disturbance configuration. The results provided in Section 5.6 are therefore the mean values over this 10 trials. In order to reduce the bias of this evaluation, we propose to, before the beginning

Coordinates	1	2	3	4	5	6	7	8	9
X	0	0.25	0.50	0.25	-0.25	0	-0.25	-0.50	0
Y	0	0.20	0	-0.20	-0.20	0	0.20	0	0
Z	-2	-2	-2	-2	-2	-2	-2	-2	-2

Table 5.1: *List of the setpoints and their coordinates (in meters).*

of each experiment protocol, stabilize the vehicle at the first setpoint. It consists in launching the control system when the vehicle is not too far from this setpoint and letting the control system regulate the vehicle over for the same amount of timesteps required during the experiments (i.e. 1000). Then, the session starts from this same setpoint. In practice, it means that there is an additional setpoint 1 in the list presented in Table 5.1, which is not taken into account when we assess the performance of the controllers. By doing that, we further minimize the bias of the evaluation as both controllers will start roughly exactly from the same state. We found this practice to be particularly relevant when we have current disturbances in the pool. In this case, without the double setpoint 1, the start of the sessions is very different due to the drift generated by the current.

The experiment task is illustrated in Figure 5.2 where the 9 setpoints are illustrated. The disturbance generator and the camera are fixed to a metallic arm that is fixed on the side of the pool at approximately 30 centimeters and 2 meters from the edge (respectively). The vehicle is asked to perform station keep at each setpoint in ascending order. The setpoints have been placed

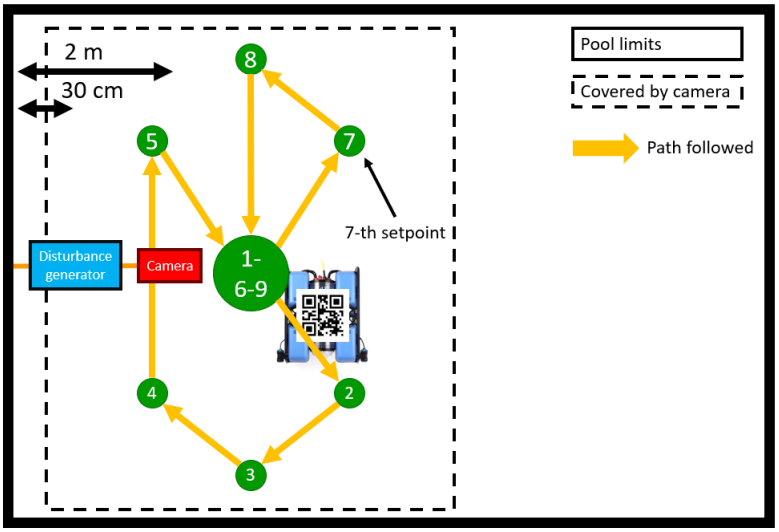
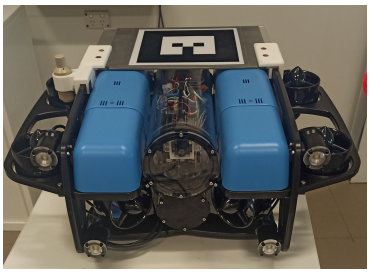


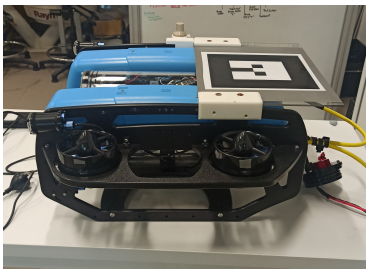
Figure 5.2: *Illustration of the station-keeping task performed for our experimental validation. The vehicle has to stabilize at 9 setpoints in space while being perturbed by an external current source.*

5.2 Pose estimation without GPS

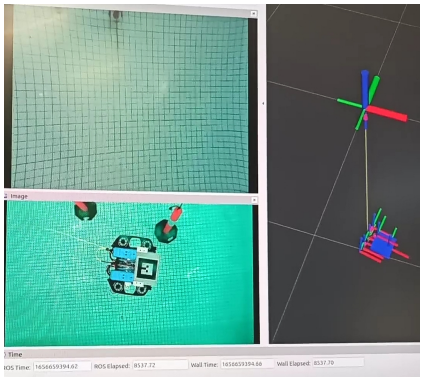
The Bluerov platform used for our experiments is equipped with an IMU and a frontal camera. No sensor allows us to directly estimate the position of the vehicle (such as a DVL). Therefore, we had to design a hybrid localization system composed of a camera and marker. The marker is placed on the top side of the vehicle while the camera is facing down toward the as illustrated in Figure 5.3. Then, a Kalman Filter uses data from the IMU and from the visual tracking system to estimate the position and orientation of the Bluerov. This solution is provided directly by the Centre for Maritime Engineering at FLINDERS University and therefore no further details will be given on its design as it is not the scope of the thesis.



(a) *Front view of the vehicle.*



(b) *Side view of the vehicle.*

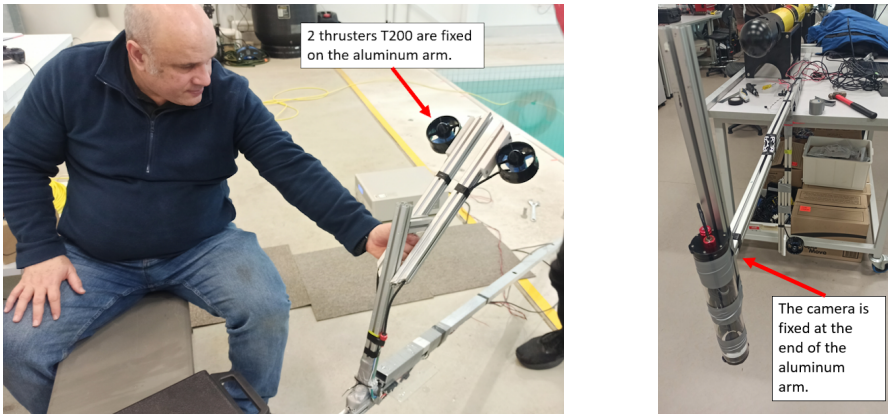


(c) *Illustration of the camera feedback (left half of the screen) and the resulting pose estimate represented by the position vectors (right half of the screen).*

Figure 5.3: *The data from sensors and the marker tracking system are fused using a Kalman filter to compute an estimation of the vehicle position.*

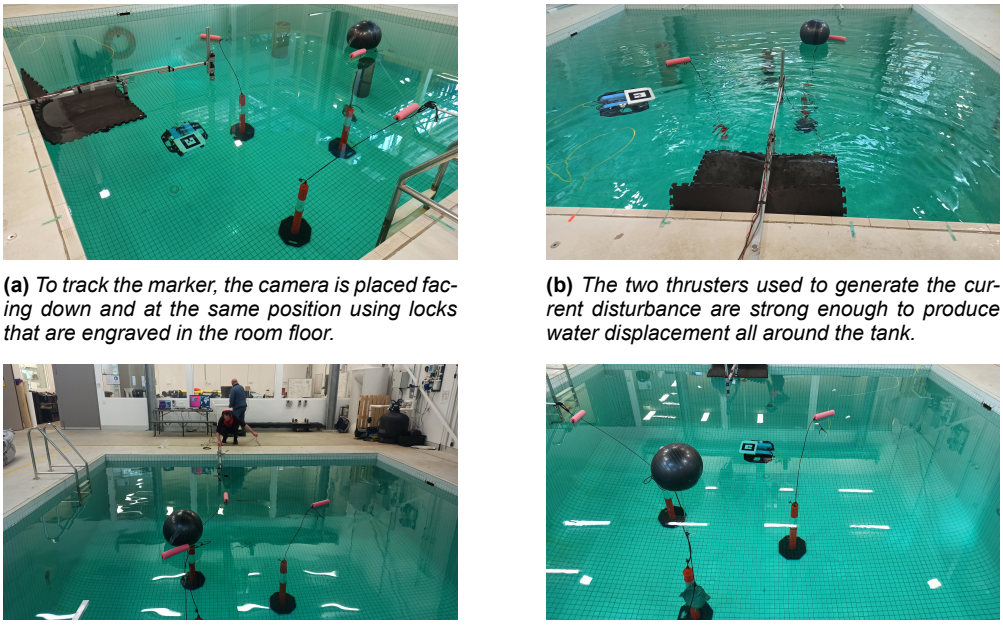
5.3 Disturbance generator

In order to evaluate the robustness of the controllers against disturbance, we proposed to create an artificial current in the water tank. To that end, we fixed two thrusters of type T200 (the same as the ones on the Bluerov platform) on the aluminum arm where the camera is attached as illustrated in Figure 5.4. We chose a particular placement and orientation of the thruster such as to optimize the field of effect in the pool. The thrusters are controlled through ESC input that we set to 1625, which according to Blue Robotics documentation gives around 8 Newtons of thrust per thruster. The total current draw for the pair is approximately 2.7A, providing a power draw of around 38 Watts.



(a) **(b)**

Figure 5.4: *Two thrusters generate current disturbance (a) while the camera is fixed at the end of the arm and is facing down (b).*



(a) *To track the marker, the camera is placed facing down and at the same position using locks that are engraved in the room floor.*

(b) *The two thrusters used to generate the current disturbance are strong enough to produce water displacement all around the tank.*

(c) *Because of the property of the different lights of the room, we had to design a new marker, covered with a layer of non-shiny material.*

(d) *The setpoints are chosen so as to always be visible by the camera, resulting in a working space of about 5m long, 3m wide, and 3m deep.*

Figure 5.5: *Illustration of the setup for the experiments.*

5.4 Design of the learning-based adaptive controller

Soft Actor-Critic with Automatic Temperature Adjustment

Motivated by the fact the we are deploying our learning-based adaptive controller on a real vehicle, and because we have access to a sub-optimal simulated model of the vehicle, we proposed to use this time the second version of the SAC algorithm (see Section 2.2.8). The main difference between these two version is that with the latter, the temperature alpha in the maximum entropy objective function Eq. (2.70). With this change, the reward scale does not needs to be tuned as the relative weight of the entropy term is adapted to satisfy a minimal entropy constraint. In fact, as studied in [Haa+18a], the reward scale influences greatly the exploration-exploitation tradeoff of maximum entropy reinforcement learning methods. The optimal value of reward scale is difficult to determine beforehand and is essentially impossible to predict for a physical system; This was illustrated with the Minitaur platform, a small-scale quadruped with eight DoFs, considered in [Haa+18a], for which two orders of magnitude larger reward scale was required to work properly (compared to environments from the OpenAI gym benchmark suite [Bro+16]). On the other hand, the target entropy Eq. (2.82) is easier to tune as it is a function of the dimension of the action space. It was proposed in [Haa+18a] that the target entropy is equal to -1 per action dimension which can be interpreted as setting the weight of the entropy in the objective function as 1 percent for a given action space of dimension n. Therefore, the resulting dual constraint optimization for the policy is defined as:

$$J(\alpha) = \mathbb{E}_{s_t \sim D, a_t \sim \pi_\mu} \big[-\alpha \log \pi_\mu(a_t|s_t) + \alpha \times 18 \big]. \tag{5.3}$$

with $\mathcal{H} = -18 = -dim(u)$ is the target entropy that is equal to the dimension of the action space which is here 18 (since we have 3 pole values per DoF and we are controlling the vehicle in the 6-DoFs 2.3.2).

State vector

At each timestep, the agent captures an observation vector o_t representing the process dynamics that we defined as:

$$o_t = [\ a_{t-1}\ ;\ \Theta\ ;\ V\ ;\ \dot{V}\ ;\ \Omega\ ;\ e_t\ ;\ e_{L2}\ ], \tag{5.4}$$

where

- $a_{t-1} \in \mathbb{R}^{18}$ are the last action estimated (i.e. poles value),
- $\Theta = [\phi; \theta; \psi]$ are the Euler orientation of the vehicle (roll, pitch and yaw respectively),
- $V = [v_x; v_y; v_z]$ and $\Omega = [\omega_\phi; \omega_\theta; \omega_\psi]$ are respectively its linear and angular velocities,
- $\dot{V} = [\dot{v_x}; \dot{v_y}; \dot{v_z}]$ is its linear acceleration,
- $e_t \in \mathbb{R}^6$ are the error values on each setpoint,
- and e_{L2} as described in Section 4.1.2,

The dimension of the observation vector o_t is therefore equal to 40. Noted that with this observation vector (5.4), the current disturbance characteristics are not included. In order to improve the process observability and following our previous results [Cha+20a], we construct our state vector s_t out of the current and past observation vectors along with their two-by-two difference. This results in a 120 dimensional state space defined as:

$$s_t = [o_t\ ;\ o_{t-1}\ ;\ o_{t-1} - o_t]. \tag{5.5}$$

Reward function

Since we are using the second version of SAC 2.2.8, the reward scale does not required to be tuned. Thus, we proposed the following reward design:

$$r(s_t) = \exp\left[-(e_{L2}(t))\right] \tag{5.6}$$

Contrary to the previous reward function designs Eqs. (3.7), (3.25), (4.7), and (4.18), the reward scale is not controlled here. This means that the reward signal (5.6) is defined in $r(s_t) = [0, 1]$.

Randomized environmental complexity as domain randomization

Despite the stability components of our learning-based adaptive controller, training directly on the real platform can not be considered due to the autonomy of the batteries. We, therefore, proposed to perform the training on a simulated version of the BlueRov platform. The resulting policy will then be transferred to the real platform. As discussed in Section 2.2.9, the distribution shift problem in reinforcement learning is mainly due to the difference in the spaces of states observed during training and evaluation. In our case here, the distribution shift arises from the fact that the policy will be trained in simulation, where the space of states is too perfect compared to what will be observed by the real robot (due to noise on sensorial data coming from imperfection, latency, power loss, and a great number of other natural phenomena). In addition, as we were able to observe in a previous work [CHA+22] (see Section 4.2), despite solving the task in training, the resulting policy can still has notably lower performance when the characteristics of the training environment greatly vary (see Section 4.2.5).

There exist many techniques to reduce the reality gap between simulation and the real world, and the most used one is denoted as *Domain Randomization* (DR). In DR, the environment that we have access to is denoted as *source domain* while the environment that we want to transfer to is denoted as *target domain*. Training is often possible only in the source domain where we have N randomization parameters that we can module in order to vary the characteristics of the domain. Thus, we can define a configuration ξ as sampled from a randomization space $\xi \in \Xi \subset \mathbb{R}^N$. During training, DR consists in collecting data from the source domain with the parameters randomization applied. By doing that, the policy is exposed to a great variety of environments and learns to generalize better than when exposed to a single environment. The randomization parameters can control the appearance of the environment such as:

- position, shape, and colors of the objects,
- texture of material,
- lighting condition,
- random noise added to images,
- or position, orientation, and field of view of the simulated camera.

These parameters can also control the physical dynamics of the domain such as:

- mass and dimension of objects,
- mass and dimension of vehicles,
- damping and friction of the joints,
- observation noise,
- or action delay.

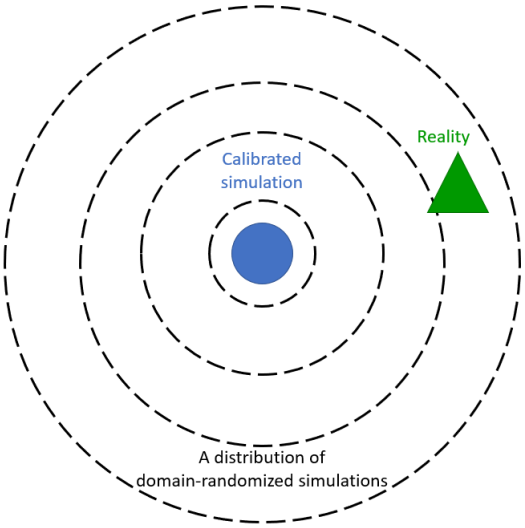


Figure 5.6: *Conceptual illustration of domain randomization. The ambition is to make the policy experience so many variations of the source domain that the target domain will only be another variation among the others.*

We proposed what we called incremental environment complexity training in a previous paper [Cha+20b] which can be seen as a variation of domain randomization. It consisted in training the agent in different variations of the environment in terms of the complexity of the task (as measured by the amount and shape of obstacles in the environment). Then, the agent transits between these domains based on its current performance as measured by the success rate. The advantage of this approach is that if the policy is not able to solve the current complexity, it will be sent back to the previous one that was solved which ensures not getting stuck in a bad regime. With the right tuning, we can ensure that the agent transit smoothly between each configuration until the last one. However, with this approach, We can not control the amount of data collected in each complexity configuration. One configuration can be way more explored than another which can lead to overfitting.

In the paper [Cha+22] presented in Section 3.2.2, we proposed another incremental environment complexity methodology where we increased over time the complexity of the task until it reaches the complexity of the target domain. We made sure this time that each configuration is explored for the same amount of timesteps to avoid overfitting. However, we can not in advance guarantee that the agent will solve each configuration. A high-complexity configuration might require more time for the agent to solve compared to a simpler one.

Following these results, and based on the performance variation observed in our paper [CHA+22] (presented in Section 4.2) we proposed for the considered process here to divide the training environment in three complexity configuration as measured by the amount of disturbance as follows:

- Configuration 1: no disturbance at all.
- Configuration 2: sea current disturbance that does not vary within the episode.
- Configuration 3: see current disturbance that change at a random time within the episode between timestep 100 and 400, out of 500. The value 500 was chosen as the maximum value for the length of the episodes in accordance with the desired settling time 5.1.

Finally, before the beginning of each episode, the agent has a probability to experience each complexity configuration P that is equal for each of them. By doing so, we are sure that each configuration will be explored uniformly (to avoid overfitting) and the hardest configuration of the environment, that is the closest one to the target domain, will be experienced very early in the training phase (to force the agent to diversify its actions). This methodology is illustrated in Figure 5.7 where the choice of complexity configuration is performed after the end of each episode.

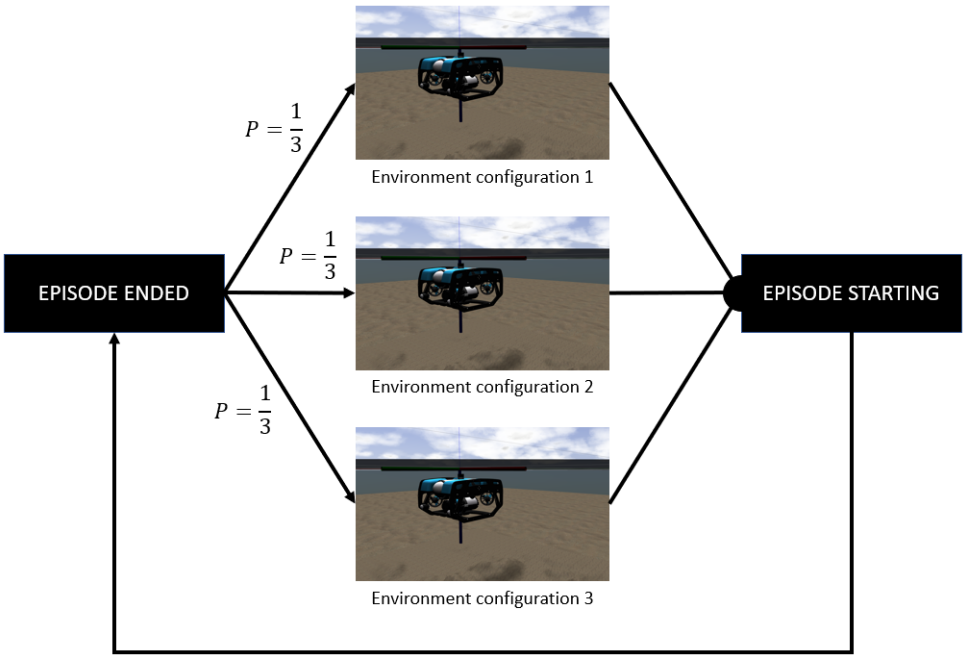


Figure 5.7: *Illustration of the proposed domain randomization method. We proposed to divide the training environment into three configurations in terms of environment complexity. At the beginning of each episode, the environment characteristics are set to one of these configurations with an equal probability P for each case.*

Exploration strategy

We used the adaptive parameter noise [Pla+18] technique (as described in Section 2.2.5) with the following values: the metric $\mathbb{E}_s[\cdot]$ is estimated over a batch of 1000 samples from the Replay Buffer, the initial standard deviation is this time equal to 1.0, the threshold is set to 0.10 and the update rate is reduced to $\alpha = 1.005$. As recommended in [Pla+18], in order to not get stuck, which can still happen with a perturbed policy, we combine the parameter noise with an ϵ-greedy policy 2.2.5 where each action holds an independent probability $\epsilon = 0.10$ to be random. During the evaluation, both parameter noise and ϵ-greedy are removed.

5.5 Training

The learning consists in performing a total of 3000 episodes of maximum timesteps set to 500 which takes approximately 4 hours (given that the training is performed in real-time factor which means that is it equivalent to 4 hours of real-life usage of the vehicle). Before an episode begins, the environment, the configuration of the environment characteristics is chosen as described in Section 5.4. The complete list of hyperparameters is provided in Table 5.2 with the details on the DRL framework.

Table 5.2: *List of hyperparameters and their values for experimental validation.*

Training hyperparameter	Value
SAC version	2 (see Section 2.2.8)
Activation function	Leaky ReLU
Optimizer (all networks)	Adam [KB15]
Learning rate (all networks)	3×10^{-4}
Discount factor (γ)	0.99
Mini-batch size	256
Target network smoothing coefficient (Δ)	0.005 (see Section 2.2.8)
Delayed update trick [FHM18]	True
Critics L2 regularization	0.001
Layer Normalization [BKH16] (all networks)	True
Automatic temperature adjustment	True
Replay buffer max size	$1e6$
Replay start size	$1e4$
Experience Replay method	BIER (see Section 3.3.3)

5.6 Results

Table 5.3: *RMSE without disturbance.*

Setpoint	Model-based	Learning-based
1	0.1189	**0.0414**
2	0.1366	**0.0509**
3	0.1673	**0.0546**
4	0.1433	**0.0544**
5	0.1085	**0.0740**
6	0.0914	**0.0498**
7	0.1380	**0.0534**
8	0.1311	**0.0463**
9	0.1106	**0.0622**

Table 5.4: *Std RMSE without disturbance.*

Setpoint	Model-based	Learning-based
1	0.0241	**0.0147**
2	0.0325	**0.0229**
3	0.0335	**0.0242**
4	0.0332	**0.0254**
5	0.0376	**0.0295**
6	0.0247	**0.0216**
7	0.0355	**0.0237**
8	0.0471	**0.0256**
9	0.0520	**0.0285**

Table 5.5: *Normalized mean $\sum |u|$ without disturbance.*

Setpoint	Model-based	Learning-based
1	0.1365	**0.1361**
2	0.1575	**0.1507**
3	0.1806	**0.1464**
4	0.1907	**0.1494**
5	**0.1036**	0.1528
6	**0.0880**	0.1526
7	**0.1209**	0.1428
8	**0.1141**	0.1408
9	**0.1289**	0.1579

Table 5.6: *Normalized mean Return without disturbance.*

Setpoint	Model-based	Learning-based
1	0.7698	**0.8992**
2	0.7517	**0.8815**
3	0.6934	**0.8740**
4	0.7185	**0.8738**
5	0.7871	**0.8390**
6	0.8336	**0.8863**
7	0.7353	**0.8769**
8	0.7617	**0.8929**
9	0.7785	**0.8595**

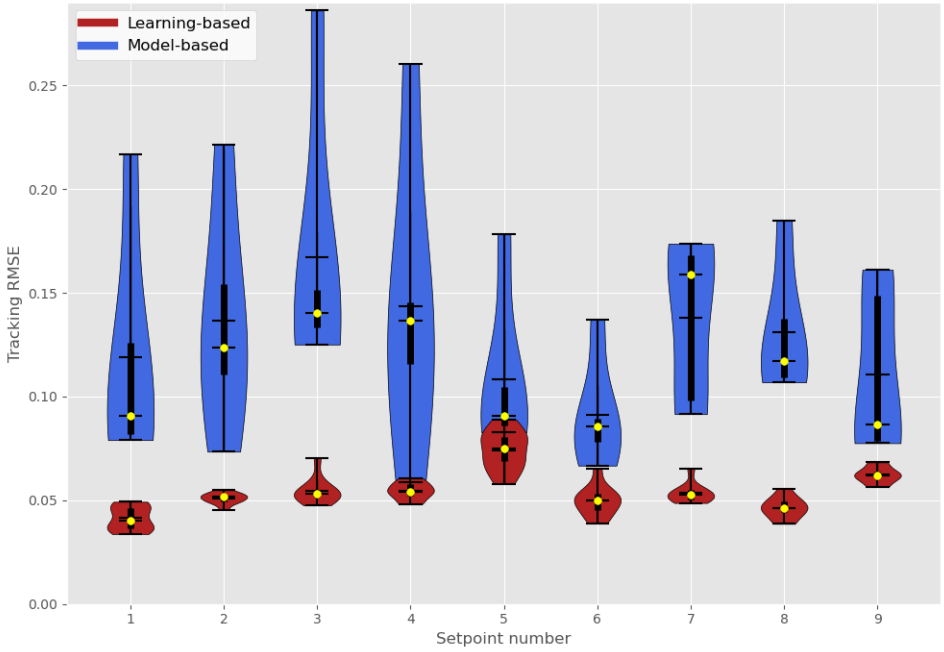


Figure 5.8: *Illustration of the experimental performance of the controllers without disturbance.*

The results provided in Tables 5.3-5.6 are associated with the experimental scenario where no current disturbance is applied to the vehicle. We can see that our learning-based (LB) controller displays better performance compared to the model-based (MB) controller. In terms of root mean squared error (RMSE) on the setpoint, the LB controller holds the smallest RMSE for every setpoint. On average, the gain in RMSE is about $\sim 241\%$ with our LB controller. When we take a look at the standard deviation (Std) of the RMSE, which can be seen as a measure of robustness, we can also observe that the LB controller is doing better than the MB controller on every setpoint. The Std of the RMSE is again on average about 2 times smaller than our method. This tendency is furthermore perceptible is the violin plots provided in Figure 5.8 where we can see the median and quartile values computed over the 10 trials. We can see that the performance of the LB controller is notably better (i.e. lower error) and way more robust than the MB controller (i.e. less disseminated). On average, the gain in Std of RMSE is about $\sim 148\%$ with our LB controller. When we take a look at the norm of the control inputs in Table 5.5, which can be seen as a measure of power consumption, we can observe a less apparent difference between the models. In fact, for the first half of the setpoint, the LB controller requires smaller control inputs to stabilize the vehicle, while it requires larger control inputs to stabilize the vehicle at the second half of setpoints. On average, the LB controller consumes 15% more energy than the MB controller. Finally, when we take a look at the return of the agent in Table 5.6, which can be used as a metric to assess if the policy is behaving as desired, we can that our LB controller is again doing better every setpoint. The normalized mean return of the LB controller is on average about 1.13 times higher than the MB controller.

Table 5.7: *RMSE with disturbance.*

Setpoint	Model-based	Learning-based
1	0.3723	**0.0882**
2	0.3587	**0.0844**
3	0.3420	**0.0731**
4	0.3645	**0.0783**
5	0.3686	**0.1120**
6	0.3647	**0.0797**
7	0.3648	**0.1149**
8	0.3606	**0.1385**
9	0.3826	**0.1100**

Table 5.8: *Std RMSE with disturbance.*

Setpoint	Model-based	Learning-based
1	0.1573	**0.0289**
2	0.0947	**0.0330**
3	0.0888	**0.0285**
4	0.1068	**0.0294**
5	0.0893	**0.0451**
6	0.1005	**0.0277**
7	0.0969	**0.0319**
8	0.0987	**0.0530**
9	0.1206	**0.0444**

Table 5.9: *Normalized mean $\sum |u|$ with disturbance.*

Setpoint	Model-based	Learning-based
1	**0.1449**	0.1816
2	**0.1407**	0.1739
3	**0.1362**	0.1407
4	**0.1406**	0.1475
5	**0.1424**	0.1637
6	**0.1417**	0.1311
7	**0.1404**	0.1389
8	**0.1424**	0.1874
9	**0.1530**	0.1717

Table 5.10: *Normalized mean Return with disturbance.*

Setpoint	Model-based	Learning-based
1	0.5260	**0.7972**
2	0.6642	**0.8008**
3	0.6648	**0.8384**
4	0.6309	**0.8348**
5	0.5326	**0.7753**
6	0.5646	**0.8309**
7	0.5461	**0.7752**
8	0.6478	**0.7486**
9	0.4831	**0.7840**

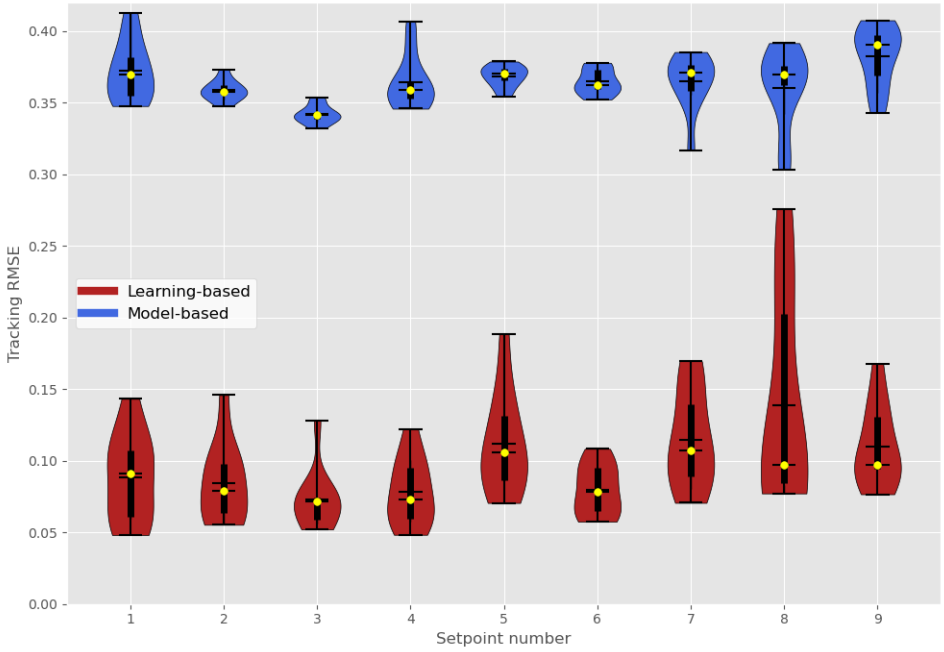


Figure 5.9: *Illustration of the experimental performance of the controllers with disturbance.*

When facing current disturbance, the benefits of our proposed method are even more prominent. This is illustrated in the results provided in Tables 5.7-5.10 is associated with the experimental scenario where we apply a current disturbance to the vehicle. Again, we can see that our LB controller displays better performance compared to the MB controller. In terms of RMSE on the setpoint, the LB controller holds the smallest RMSE for every setpoint. On average, the gain in performance in terms of setpoint regularization is about $\sim 207\%$ with our LB controller. In terms of the Std of the RMSE, the LB controller is also doing better than the MB controller on every setpoint. As illustrated in Figure 5.9 where we can see the median and quartile values computed over the 10 trials. We can see that the performance of the LB controller is notably better (i.e. lower error) and way more robust than the MB controller (i.e. less disseminated). On average, the gain in Std of RMSE is about $\sim 314\%$ with our LB controller. When we take a look at the norm of the control inputs in Table 5.5, which can be seen as a measure of power consumption, we can observe a less apparent difference between the models. In fact, for the first half of the setpoint, the LB controller requires smaller control inputs to stabilize the vehicle, while it requires larger control inputs to stabilize the vehicle at the second half of setpoints. On average, the LB controller consumes 9% more energy than the MB controller. Finally, when we take a look at the return of the agent in Table 5.6, which can be used as a metric to assess if the policy is behaving as desired, we can that our LB controller is again doing better on every setpoint. The normalized mean return of the LB controller is on average about 1.38 times higher than the MB controller.

In Figures 5.10a to 5.10d, we provide violin plots of the results. On the top row, we plot the performance of both controller without disturbance (left) and against current disturbance (right). On the bottom row, we plot the performance displayed without disturbance and against current disturbance by the model-based controller (left) and by the proposed learning-based adaptive controller (right). We can see that despite using a sub-optimal simulated model of the AUV, the policy performed notably better when transfer on the real platform.

In Figures 5.11a to 5.16b, we plot the evolution of each DoF during a trial under current disturbance. We can see that the proposed learning-based adaptive controller displays lower overshoot and is better at tracking the desired trajectory.

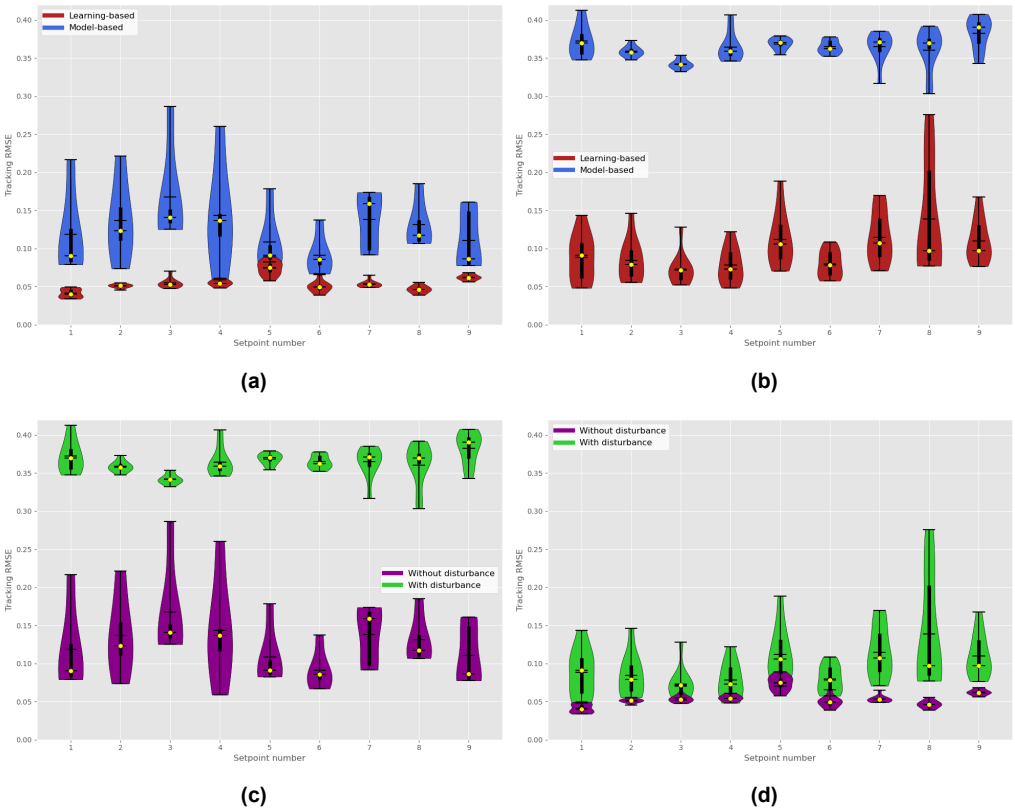
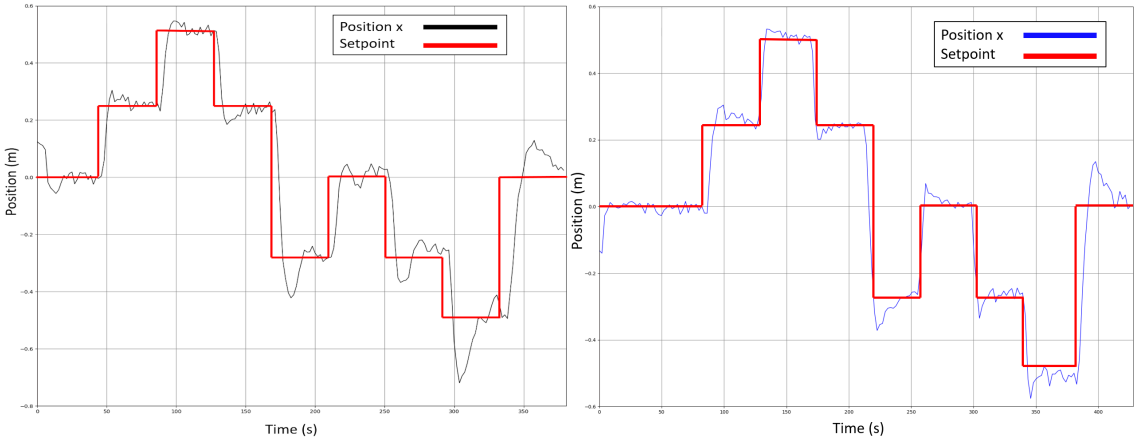
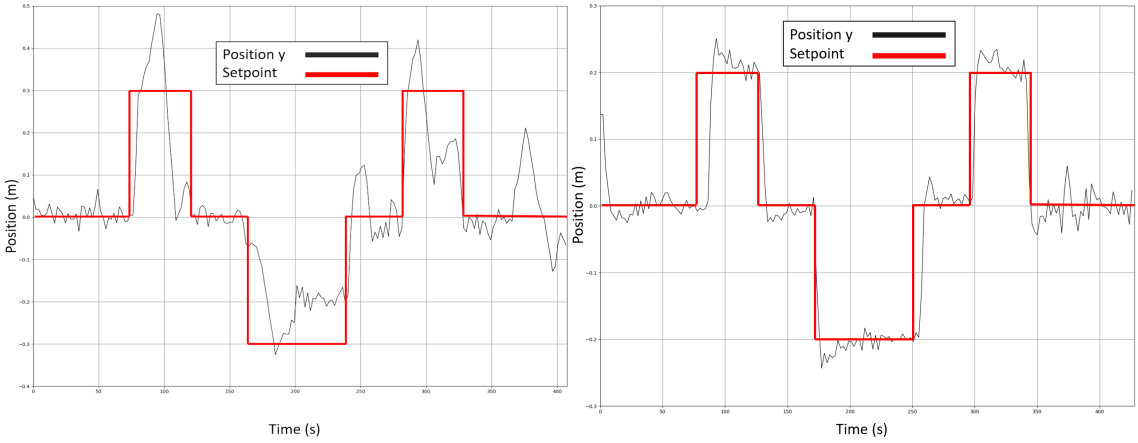


Figure 5.10: *Illustration of the experimental performance (the complete results are provided in Tables 5.3-5.10). In the first row, we present the performance of both controllers without disturbance in Figure 5.10a, and against current disturbance in Figure 5.10b. On the second row, we display the performance of each controller against both operating conditions with the performance of the model-based controller in Figure 5.10c and the learning-based controller in Figure 5.10d.*



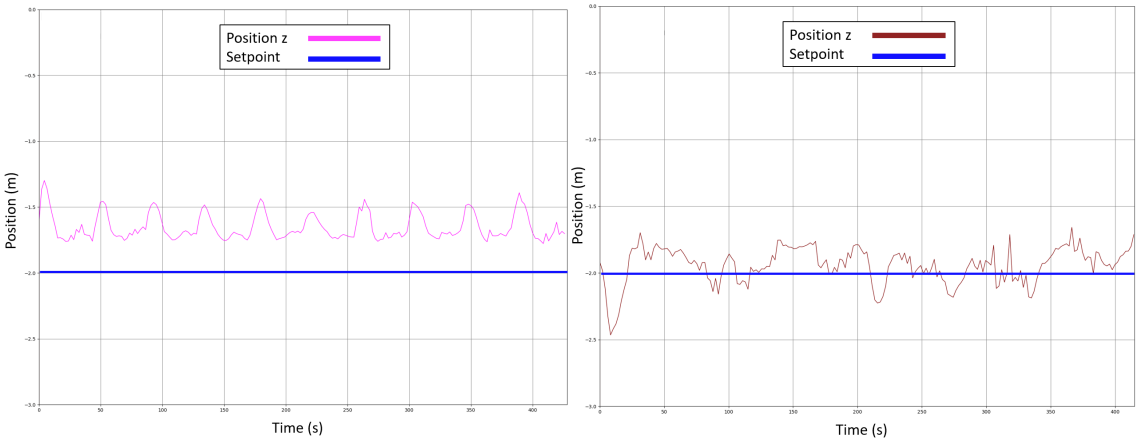
(a) **(b)**

Figure 5.11: *Evolution of the position X with the MB controller (a) and with the LB controller (b)*



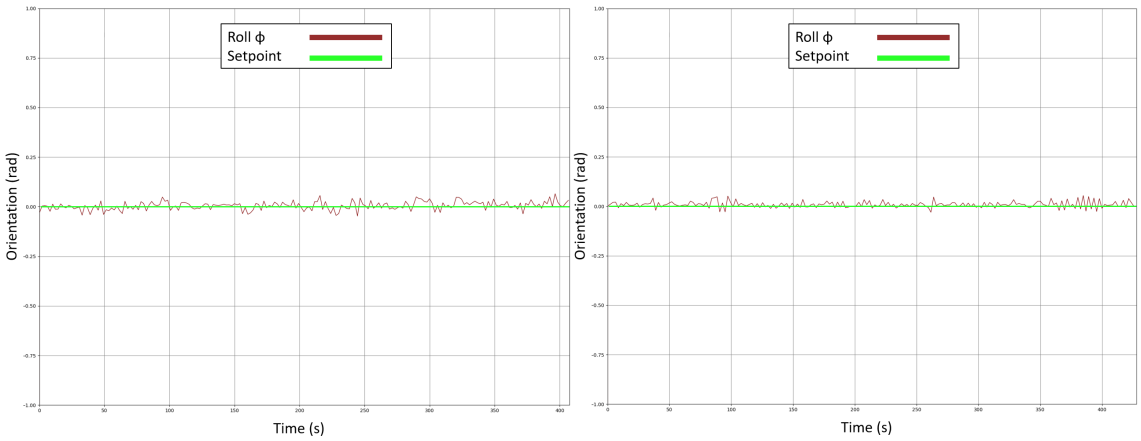
(a) **(b)**

Figure 5.12: *Evolution of the position Y with the MB controller (a) and with the LB controller (b)*



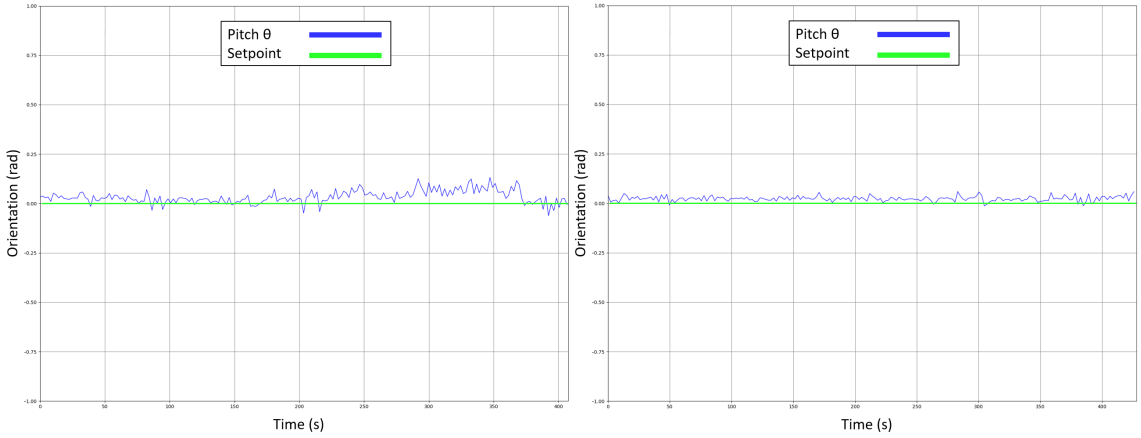
(a) **(b)**

Figure 5.13: *Evolution of the position Z with the MB controller (a) and with the LB controller (b)*



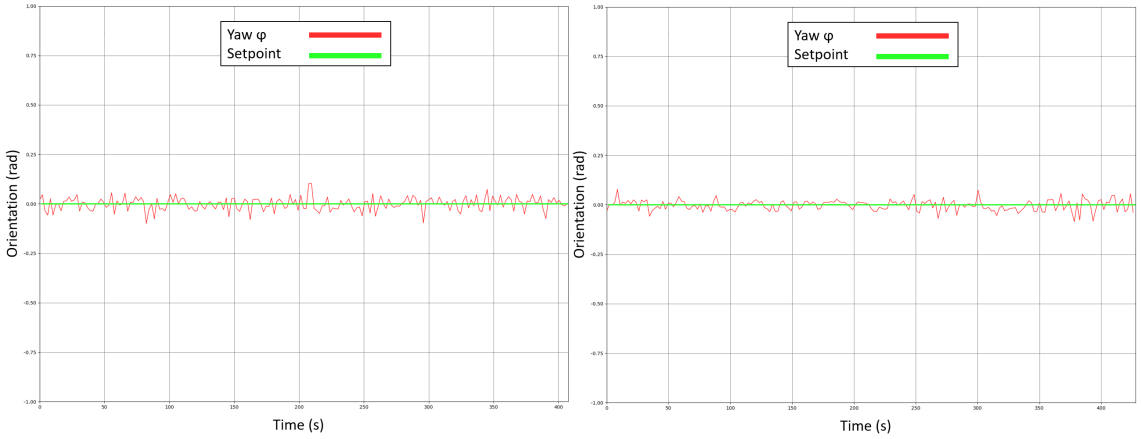
(a) **(b)**

Figure 5.14: *Evolution of the roll ψ with the MB controller (a) and with the LB controller (b)*



(a) **(b)**

Figure 5.15: *Evolution of the pitch θ with the MB controller (a) and with the LB controller (b)*



(a) **(b)**

Figure 5.16: *Evolution of the yaw ϕ with the MB controller (a) and with the LB controller (b)*

5.7 Summary

In this section we presented the experimental validation of the proposed learning-based adaptive control system. In order to successfully transfer the policy from simulation to the real world, we proposed a domain randomization strategy where the agent faces different levels of environment complexity as measured by the amplitude and variation of the sea current disturbance. In addition, we proposed this time to use the second version of the SAC algorithm, with the automatic adjustment of the temperature parameter (see Section 2.2.8). This way, we do not need an intensive and empirical tuning of the reward scale parameter anymore. We compare our method to a purely model-based version of the exact same control structure that was proposed in [WS18]. The results show that the learning-based adaptive controller is doing better at stabilizing the AUV in both operating conditions: without and with sea current disturbance.

6 General conclusions and perspectives

Contributions summary

In this thesis we proposed a learning-based approach to the problem of adaptive control for autonomous underwater vehicles, with a focus on external disturbances. Following an in-depth preliminary investigation (Section 3), we identified a set of key problems which make the problem challenging. We now summarized these topics and specify how we tackled them in the learning-based adaptive control system in Section 3.3 and how we evaluate it under simulation and in the real world as presented in Section 4 and 5 respectively.

Adjustment mechanism: In the PID control framework, the control parameters are the gains of the control law. These variables can take numerous values and we have discussed why in our case, classic model-based tuning methods [Wan05] can not be considered and why it is difficult to define the bounds of this space. In addition, control performance requirements are not straightforward to derive in the space of gains. In Section 3.3.1, this space of gains is transformed into the space of poles where only desired values are considered (i.e. poles of null imaginary part and negative real part) and where it is easier to impose bounds on the poles to meet predefined control performance requirements (e.g. desired settling time or overshoot). To achieve that, we introduced a novel mapping for the PID control law allowing us to switch from one domain to the other. This enabled us to bound the space of poles with respect to the desired control performance, which is not possible in the space of gains. Finally, we considered maximum entropy deep reinforcement learning to explore this space of desired values, allowing us to adapt the controller response to process variations (because the control parameters are a nonlinear function of a state representation of the process) and to guarantee some stability components on the system despite the use of deep neural networks. In fact, the proposed pole-placement design ensures that the poles hold the Routh–Hurwitz stability criterion and we have seen in Section 3.2.3 that despite not taking into account Lyapunov stability components in the learning procedure, the resulting policy is matching the stability of a purely Lyapunov-based adaptive controller. We have been able to study and show that this combination, compared to a completely DRL-based model-free controller, is better at controlling the AUV and rejecting current disturbance. The proposed strategy can be used whenever a feedback controller can be considered, that is the case of most closed-loop systems.

Unobservability of the disturbance: The sea current disturbance that we want to compensate for is not observed by the control system. In other words, the value of the current velocity and orientation is not included in the state vector of the process. Nevertheless, we cope with this limitation by augmenting the state vector in such a manner that the effect of the disturbance can still be noticed. In practical terms, even without measuring the current directly, we can observe changes in current by observing the state of the vehicles. The state vector incorporates variables such as the AUV linear and angular velocities, on which the current disturbance has a direct impact. In Section 4.2.5, we show that despite not measuring it explicitly, with this procedure the variation in current disturbance is still detected and is compensated by fast adjustment of the pole values.

Sample efficiency: Deep policy gradient methods require a great number of interactions with the environment to converge to a satisfying behavior. This can be explained by the fact that these methods do not learn what a good action is, but rather which action is better than the others. For this reason, these methods need to experience a large number of interactions to understand which set of actions is really the optimal one for the long-term return objective. We have been able to study how various variables of deep policy gradient methods can influence their sample efficiency which can be defined as: 1) the number of gradient steps before reaching performance convergence and 2) the standard deviation of the agent performance. Among the existing variables, we proposed to study the Experience Replay mechanism following numerous studies from the literature [ZS17][Sch+16][Fed+20] and [Hay+21] which emphasize the impact of this technique. In Section 3.3.3 we proposed a novel Bio-Inspired Experience Replay allowing us to train deep policy gradient methods faster and with improved stability. It consists in adding elements from the biological replay mechanism [Hay+21] into our deep learning systems. BIER can be applied to any reward-based MDP.

Sim-to-real transfer: The transfer of a policy trained under simulation into the real world is difficult due to the reality gap between simulators and the operating environment. There exist numerous techniques to facilitate this procedure, but in our case, since we are transferring control parameters, environment complexity randomization was proposed. We first propose this method in [Cha+20b] which consists in dividing the training environment into sub-environments of the increasing complexity of the task. Then during training, the agent transit between the randomized environment based on its current performance. For the experimental validation of our method, we proposed an improved design of this procedure where the agent holds the same probability to experience each configuration of environment complexity. We showed that what is important is

the variety of the samples and that the agent can directly face the most difficult configuration of the environment early in training without damaging its long-term performance. Moreover, we showed that transferring control parameters is easier than transferring control inputs because we do not learn the dynamics of a specific system and environment but rather we learn how to adapt to the variation of a process: from this point of view, the difference between two vehicles will be how the errors on the setpoint vary. This was highlighted in Section 5 where despite training the policy using a model of the real vehicle that is far from optimal (i.e. the physical parameters of the simulated model do not exactly match the behavior of the real vehicle), the resulting policy was able to control effectively the real vehicle, even better than a purely nonlinear model-based controller.

Perspectives

Following the study, the design, and the simulated and experimental validation of the proposed learning-based adaptive control system, I have been able to identify different perspectives to improve these steps. In this section, I will present from a more personal point of view, and discuss what is blocking the usage of these types of control systems on real platforms and in industrial contexts.

Beyond PID control: In this thesis I have been focusing on the PID control law. This choice was motivated by the fact that the AUVs of interest are fully controllable and in this case, a PID control law can ensure convergence to the steady state when time goes to infinity. Another reason, which might be the most important one, is that we do not have a reasonably good model of AUV processes to derive such optimal designs. Nevertheless, in theory, our methodology can be applied to any type of feedback control law, including: LQR [Arg+13], H_2/H_∞ PID [ACP06], L_2-gain control [FE12], MPC [Ers+21], Gain Scheduling [Cle+02], …, etc. The main difference with these control laws would be the number of control parameters and the mathematical framework used to compute the resulting control inputs (e.g. the control inputs are the results of an optimization problem with H_∞ PID controller and not derived from a linear function as with a standard PID controller). The methodology would remain the same, that is to use a deep reinforcement learning algorithm to adjust these parameters to process variations. A first improvement perspective would be to consider instead of the PID law, a nonlinear control law that would provide a better regularization ability (at the cost of being more expensive to compute compared to a PID controller) and with additional stability components. Lyapunov stability component can be directly incorporated in the design of the control structure when using for example L_2-gain control [EWB11] or MPC [SSB18]. By using such nonlinear control structures, the deployment of learning-based adaptive control systems on real platforms could be more easily received by both academia and the industry as we could in addition to gaining in control performance, guarantee the safety of the systems, similarly to the classic model-based methods but without a complete model of the process.

Learning how to learn: After these past years of experience in DRL, I have been able to feel how sensitive these algorithms are. There are so many parameters to tune that even for the simplest DRL algorithm, it can take several months for a system designer to build an agent that generalizes well [Gho+21]. Among the various variables that we can control, in this thesis I have been most interested in the Experience Replay (see Section 2.2.6). I decided to study its impact because I strongly believe that if we manipulate the agent's past experience in a more clever manner, we can influence the future data collection of the agent leading to higher quality interactions and thus faster learning. Despite having been pointed by the leaders of the field, like Richard S. Sutton [ZS17], Sergey Levine [Men+20], Yoshua Bengio [Fed+20], or David Silver [Sch+16], this issue remains pretty much ignored by the community with replay mechanism in our deep learning systems that are still missing important biological elements [Hay+21]. In this thesis, I tried to study this challenge by proposing the BIER method which is a first attempt at incorporating these biological elements in deep reinforcement learning. With BIER, we considered a good interaction as one associated with a reward value higher than the current mean reward value and then use these samples more often in the mini-batch gradient descent procedure. This choice of criterion is of course far from optimal. The distribution of reward can take various shapes: depending on the reward function itself (e.g. if the reward function is symmetric or not, the mean value can hold a different nature), depending on the stage of training (i.e. at the beginning of training we can expect a reward distribution that is more uniformed compare to the one observed by the end of training), depending on the task itself (e.g. with the classic mountain car problem [PL19] where the only way for the agent to reach the goal is to take initial actions that prevent it from reaching the goal), or quite simply depending on the presence or not of rewards. A direct improvement would be to adapt the criterion to the shape of the reward distribution, for example: if the current reward distribution is more likely a Gaussian distribution, the mean or even the third quantile can be considered as a threshold to define outliers, on the other hand, if the current reward distribution is more likely a Pearson distribution, based on how skewed it is we might consider different criterion to define outliers. The value of the reward is not the only reason why we would be more selective in the past experience selection. The policy gradient formulation Eq. (2.45) tells us that these type of solution methods does not learn what a good action is, but rather learns which action is better compared to the others. Accordingly, we could fairly ask ourselves: if only considering a sample of

experience associated with a Q-Value higher than the mean one will induce a positive change (with respect to the RL objective that is to maximize future return) in the policy, why should we be concerned by any other samples? In fact, since we are only interested in the maximum value of the Q-Value function (as discussed in Section 2.2.9), why would we be interested in the other parts of the function? Another perspective, or domain of interest, would be to study if it is still possible to learn the task by considering for example storing in the Replay Buffer, only the samples that meet this value criterion. This way, we would ensure that each gradient update will induce a policy improvement in terms of return, leading to faster learning. However, from an optimization point of view, using such few and particular samples to optimize a nonlinear function approximator can be very dangerous and lead to Q-Value overestimation or overfitting.

Real life reinforcement learning: In theory, the optimal procedure would consist in training deep policy gradient methods directly in the real world, removing the distribution shit problem of reinforcement learning. In practice, using today's DRL methods in the real world can not be considered for numerous reasons [Dul+20]. The principal problem is that they need a large number of interactions to reach a satisfying performance, which is also computationally expensive, and before reaching that stage the agent behavior can be dangerous for the platforms and operators. An improvement perspective would be to perform what is denoted as off-line reinforcement learning and then adjust (to a smaller extent) the solution obtained from this first phase by online reinforcement learning during operation. Off-line reinforcement learning consists in training the agent on a finite set of experiences until the convergence of the policy. This set can be obtained for instance in our case, by running a model-based controller, i.e. OFP controller, on the AUV for a short amount of time and recording the resulting s_t, a_t, r_t, s_{t+1}. Off-line reinforcement learning can be seen as a variation of imitation learning [Ras+21] in the sense that we will not be able to learn a policy better than the one used to generate the samples. However, this will be a way better starting point compared to a random policy, because the OFP controller despite not being optimal, ensures some minimal control performance and stability components. The proposed learning-based adaptive control system would then be used during operation with a limited update ability, e.g. with a lower learning rate and a lower gradient update rate. In the long run, we will need to build machines that can directly learn from real data, because that is the only way that we will get them to improve perpetually. If not, if they have to rely on simulated data, eventually the simulator becomes the bottleneck.

Simulation is not a substitute for being able to not utilize real experience: Another strategy to leverage deep policy gradient methods would be to close the reality gap between the simulator and the physical world. In this thesis, we used solely Gazebo as the simulation framework. It allowed us to simulate underwater processes and several disturbances, but it remains far from the complexity of real sea dynamics. On the other hand, trying to have an overly realistic simulation does not sound feasible as the number of particles and their interactions to model grows exponentially in the underwater context. Running such simulations will not allow us to use effectively the methods presented in this thesis. Another point of view is to augment the simulation so as to make the set of simulated states closer to the one encountered during the operation. One possible way to do that is to extract from real-life data what is truly representative of these interactions, to then use this knowledge as a model to adapt the simulated interactions to look more like real interactions, and one technology that can be used to achieve this are Autoencoders [Vin+10] [Vin+08]. Autoencoders are a very specific type of deep neural network used for unsupervised learning. It learns codings of unlabeled data by attempting to regenerate the input from the encoding. This procedure can be seen as a learning-based version of Principal Component Analysis [WEG20] which aims at reducing the dimension of the representation space. This strategy is already used on dynamical systems such as the Lorenz, Rossler, and Lotka-Volterra systems [Bak+22] where a deep autoencoder network is used to learn a coordinate transformation from the delay embedded space into a new space where it is possible to represent the dynamics in a sparse, closed form. In our case, we could use Autoencoders to learn a function that maps the state s_t that incorporates the past action performed (input of the encoder) to the next state s_{t+1} of the same size (output of the decoder). By doing that, we essentially force the model to learn a low-dimensionality representation of the dynamics of the real world. Then, we could use the resulting optimized Autoencoders to adapt the simulation by feeding as input the simulated s_t and observing as output the augmented state z_t that will be transformed such as to be more realistic. Then, the vector z_t will be used as input to the policy network instead of s_t. The objective is to train on samples that are close to the ones observed in real life despite using a simulator. This would allow, in theory, to simulate an infinite amount of realistic data to train the deep policy gradient methods and thus reduce the distribution shift of reinforcement learning. Nevertheless, real-life experiments in more complex conditions, compared to the pool experiments presented in Section 5, will have to be conducted to further validate the relevance of this approach.

External and internal disturbance rejection: In this thesis, I initially made the assumption that the AUVs of interest are fully controllable. In other words, this means that we consider the vehicle to always operate as requested and as planned. The scope of the thesis was thus defined as designing methods to detect and compensate for the external disturbance taking the form of sea-induced disturbing forces. However, in practice, the vehicle behavior will not always be as expected, and sometimes even far from it because of

power loss of the thruster or body wrench disturbance. These types of disturbance are in contrary internal to the AUV itself. They directly come from the component of the vehicles. An improvement of the method proposed in this thesis would be to take into account both external and internal disturbances. Contrary to the external disturbance, the internal ones can not be compensated without some a priori knowledge. We can not detect a deviation from normal behavior if we do not have a model of its expected behavior. This model can then either be known beforehand, or estimated using some parametric system identification techniques, including but not restricted to: Extended Kalman Filter, Least Square, Best Linear Unbiased Estimate, Maximum Likelihood Estimate, or neural networks. An intuitive methodology would be to use deep learning to perform this task [20]. The resulting estimation would then be incorporated into the proposed learning-based adaptive control system to compensate for both external and internal disturbances.

Learning to adapt to process variation, and not to a particular task or system: Across the different studies realized in this thesis, I was able to apply the learning-based adaptive control system to different types of vehicles (AUV and MAV), under different nature of disturbances (sea current disturbance and wind gust disturbance), and for different class of task (target rallying, outputs regularization, and station keeping). In all cases, the exact same control system was used and was able after learning to reach satisfying performance. An open question that remains is whether or not it would be possible for such a control system trained for example on a target rallying mission, to be used on an outputs regularization task and maintain its performance. Similarly, could we train on an AUV and transfer successfully the resulting policy to a MAV? My opinion is that it could be possible. In fact, in our learning-based design, the neural network takes as input a state representation of the process and estimate the associated poles of the PID control law. We are not learning to map one particular system as the variables in the state vector are essentially the same (i.e. IMU feedback). Whether we are trying to perform target rallying, outputs regularization, or station keeping, from a control objective point of view, the only difference is that in some cases the setpoint is a constant or a function of time. Therefore, the inputs of the control structure, that is the PID controller, remain the same: the error on the setpoint.

Bibliography

[Leb22] M. Leblanc. *Sur l'electrification des chemins de fer au moyen de courants alternatifs de frequence elevee*. 1922.

[Bel52] Richard Bellman. "On the Theory of Dynamic Programming." In: *Proceedings of the National Academy of Sciences of the United States of America* 38 8 (1952), pp. 716–9.

[Kál58] Róbert Kálmán. "Design of a Self-Optimizing Control System". In: *Journal of Fluids Engineering* (1958).

[MBT59] Eli A. Mishkin, Ludwig Braun, and John G. Truxal. "Adaptive control systems". In: 1959.

[OWK61] P.V. Osburn, H.P. Whitaker, and A. Kezer. "New developments in the design of adaptive control systems". In: *Institute of Aeronautical Sciences* (1961), pp. 61–39.

[PB61] E. S. Page and R. Bellman. "Adaptive Control Processes: A Guided Tour." In: 1961.

[Has70] W. K. Hastings. "Monte Carlo Sampling Methods Using Markov Chains and Their Applications". In: *Biometrika* 57 (1970), pp. 97–109.

[KS72] Huibert Kwakernaak and Raphael Sivan. "Linear Optimal Control Systems". In: 1972.

[Kre73] A. Krener. "On the Equivalence of Control Systems and the Linearization of Nonlinear Systems". In: *Siam Journal on Control* 11 (1973), pp. 670–676.

[Käl+79] C. Källström et al. "Adaptive autopilots for tankers". In: *Autom.* 15 (1979), pp. 241–254.

[Mor79] A. Morse. "Global stability of parameter-adaptive control systems". In: 1979.

[NV80] K. Narendra and L. Valavani. "A comparison of Lyapunov and hyperstability approaches to adaptive control of continuous systems". In: *IEEE Transactions on Automatic Control* 25 (1980), pp. 243–247.

[Ste80] G. Stein. "ADAPTIVE FLIGHT CONTROL: A PRAGMATIC VIEW". In: 1980.

[Par81] P. Parks. "Stability and convergence of adaptive controllers-continuous systems". In: 1981.

[Lan84] Y. D. Landau. "Adaptive control: The model reference approach". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-14 (1984), pp. 169–170.

[GM87] G. Goodwin and D. Mayne. "A parameter estimation perspective of continuous time model reference adaptive control". In: *Autom.* 23 (1987), pp. 57–70.

[Dav90] L. Davis. "Handbook Of Genetic Algorithms". In: 1990.

[Fos94] T. Fossen. "Nonlinear Modelling And Control Of Underwater Vehicles". PhD thesis. NUST, 1994.

[Put94] Martin L. Puterman. "Markov Decision Processes: Discrete Stochastic Dynamic Programming". In: *Wiley Series in Probability and Statistics*. 1994.

[Tes94] Gerald Tesauro. "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play". In: *Neural Computation* 6 (1994), pp. 215–219.

[Ber95] Dimitri P. Bertsekas. "Dynamic Programming and Optimal Control". In: 1995.

[BT96] D. Bertsekas and J. Tsitsiklis. "Neuro-Dynamic Programming". In: *Encyclopedia of Machine Learning*. 1996.

[CG96] M. Chilali and P. Gahinet. "H_∞ Design with pole placement constraints: an LMI approach". In: *IEEE Transactions on Automatic Control* 41 (1996), pp. 358–367.

[Doy96] John Doyle. "Robust and optimal control". In: *Proceedings of 35th IEEE Conference on Decision and Control* 2 (1996), 1595–1598 vol.2.

[VKM96] Z. Vukic, L. Kuljaca, and D. Milinovic. "Predictive gain scheduling autopilot for ships". In: *Proceedings of 8th Mediterranean Electrotechnical Conference on Industrial Applications in Power Systems, Computer Science and Telecommunications (MELECON 96)*. Vol. 2. 1996, 1133–1136 vol.2. DOI: 10.1109/MELCON.1996.551407.

[Sut+99] Richard S. Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *NIPS*. 1999.

[KW00] M. Krstić and Hsin-Hsiung Wang. "Stability of extremum seeking feedback for general nonlinear dynamic systems". In: *Autom.* 36 (2000), pp. 595–601.

[Rot00] M. Rotea. "Analysis of multivariable extremum seeking algorithms". In: *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)* 1 (2000), 433–437 vol.1.

[Yuh00] Junku Yuh. "Design and Control of Autonomous Underwater Robots: A Survey". In: *Autonomous Robots* 8 (2000), pp. 7–24.

[BR01] A. Bacciotti and L. Rosier. "Liapunov functions and stability in control theory". In: 2001.

[Spo+01] Jeffrey T. Spooner et al. "Stable Adaptive Control and Estimation for Nonlinear Systems". In: 2001.

[Cle+02] Benoit Clement et al. "AEROSPACE LAUNCH VEHICLE CONTROL: A GAIN SCHEDULING APPROACH". In: *Control Engineering Practice* 13 (2002), pp. 333–347.

[Ng+03] A. Ng et al. "Autonomous Helicopter Flight via Reinforcement Learning". In: *NIPS*. 2003.

[DL04] G. DeJong and A. Laud. "Theory and application of reward shaping in reinforcement learning". In: 2004.

[KH04] Nathan P. Koenig and Andrew Howard. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)* 3 (2004), 2149–2154 vol.3.

[Lin04] Longxin Lin. “Self-improving reactive agents based on reinforcement learning, planning and teaching”. In: *Machine Learning* 8 (2004), pp. 293–321.

[Ng+04] A. Ng et al. “Autonomous Inverted Helicopter Flight via Reinforcement Learning”. In: *ISER*. 2004.

[RC04] Christian P. Robert and George Casella. “Monte Carlo Statistical Methods”. In: *Springer Texts in Statistics*. 2004.

[Wil04] Ronald J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning* 8 (2004), pp. 229–256.

[And05] B. Anderson. “Failures of adaptive control theory and their resolution”. In: *Commun. Inf. Syst.* 5 (2005), pp. 1–20.

[AC05] B. Anderson and J. Crowell. “Workhorse AUV – A cost-sensible new Autonomous Underwater Vehicle for Surveys/Soundings, Search & Rescue, and Research”. In: *Proceedings of OCEANS 2005 MTS/IEEE* (2005), pp. 1–6.

[KK05] N. Killingsworth and M. Krstić. “PID Tuning Using Extremum Seeking”. In: 2005.

[Lib05] Daniel Liberzon. “Review of Liapunov functions and stability in control theory”. In: *Automatica* 41 (2005).

[Wan05] Qing-Guo Wang. “Handbook of PI and PID Controller Tuning Rules, Aidan O’Dwyer, Imperial College Press, London, 375pp, ISBN 1-86094-342-X, 2003”. In: *Autom.* 41.2 (2005), pp. 355–356.

[WD05] Fen Wu and Ke Dong. “Gain-scheduling control of LFT systems using parameter-dependent Lyapunov functions”. In: *Proceedings of the 2005, American Control Conference, 2005.* 2005, 587–592 vol. 1. DOI: 10.1109/ACC.2005.1470020.

[ACP06] Denis Arzelier, Benoit Clement, and Dimitri Peaucelle. “Multi-objective H2 / H∞ / Impulse-to-Peak Control of a Space Launch Vehicle”. In: *Eur. J. Control* 12 (2006), pp. 57–70.

[KK06] N. Killingsworth and M. Krstić. “PID tuning using extremum seeking: online, model-free performance optimization”. In: *IEEE Control Systems* 26 (2006), pp. 70–79.

[KRP06] Leszek Koszalka, Radoslaw Rudek, and Iwona Pozniak-Koszalka. “An Idea of Using Reinforcement Learning in Adaptive Control Systems”. In: *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies* (2006), pp. 190–190.

[Wan+06] Cong Wang et al. “An ISS-modular approach for adaptive neural control of pure-feedback systems”. In: *Autom.* 42 (2006), pp. 723–731.

[Gol08] Goldberg. “Genetic Algorithms”. In: 2008.

[Nic+08] John R. Nickolls et al. “Scalable parallel programming with CUDA”. In: *2008 IEEE Hot Chips 20 Symposium (HCS)* (2008), pp. 1–2.

[Vin+08] Pascal Vincent et al. “Extracting and composing robust features with denoising autoencoders”. In: *ICML ’08*. 2008.

[Wie+08] Daan Wierstra et al. “Natural Evolution Strategies”. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)* (2008), pp. 3381–3387.

[MCY09] G. Marani, S. Choi, and J. Yuh. “Underwater autonomous manipulation for intervention missions AUVs”. In: *Ocean Engineering* 36 (2009), pp. 15–23.

[Nes09] D. Nesic. “Extremum seeking control: Convergence analysis”. In: *2009 European Control Conference (ECC)* (2009), pp. 1702–1715.

[Qui09] Morgan Quigley. “ROS: an open-source Robot Operating System”. In: *ICRA 2009*. 2009.

[Sut+09] Richard S. Sutton et al. “Fast gradient-descent methods for temporal-difference learning with linear function approximation”. In: *ICML ’09*. 2009.

[WH09] Cong Wang and David. J. Hill. “Deterministic Learning Theory for Identification, Recognition, and Control”. In: 2009.

[DAL10] Z. T. Dydek, A. Annaswamy, and E. Lavretsky. “Adaptive Control and the NASA X-15-3 Flight Revisited”. In: *IEEE Control Systems* 30 (2010), pp. 32–48.

[Vin+10] Pascal Vincent et al. “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion”. In: *J. Mach. Learn. Res.* 11 (2010), pp. 3371–3408.

[Ber+11] Sylvain Bertrand et al. “A hierarchical controller for miniature VTOL UAVs: Design and stability analysis using singular perturbation theory”. In: *Control Engineering Practice* 19 (2011), pp. 1099–1108.

[EWB11] Mohsen M. Ezzeldin, Siep Weiland, and Paul P. J. van den Bosch. “Robust □2 control for a class of nonlinear systems: A parameter varying Lyapunov function approach”. In: *2011 19th Mediterranean Conference on Control & Automation (MED)* (2011), pp. 213–218.

[HHL11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration”. In: *Learning and Intelligent Optimization*. Ed. by Carlos A. Coello Coello. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 507–523. ISBN: 978-3-642-25566-3.

[Joy11] James M. Joyce. “Kullback-Leibler Divergence”. In: *International Encyclopedia of Statistical Science*. 2011.

[SM11] Richard S. Sutton and Hamid Reza Maei. “Gradient temporal-difference learning algorithms”. In: 2011.

[Ber12] V. Berg. “Development and Commissioning of a DP system for ROV SF 30k”. PhD thesis. NTNU, 2012.

[DWS12] Thomas Degris, Martha White, and Richard S. Sutton. “Off-Policy Actor-Critic”. In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML’12. Edinburgh, Scotland: Omnipress, 2012, pp. 179–186. ISBN: 9781450312851.

[DK12] Sam Devlin and D. Kudenko. “Dynamic potential-based reward shaping”. In: *AAMAS*. 2012.

[FE12] Sami El Ferik and M. Emzir. “PCH-Based □2 Disturbance Attenuation and Control of Autonomous Underwater Vehicle”. In: *IFAC Proceedings Volumes* 45 (2012), pp. 192–197.

[GNO12] B. Gilmour, G. Niccum, and T. O’Donnell. “Field resident AUV systems — Chevron’s long-term goal for AUV development”. In: *2012 IEEE/OES Autonomous Underwater Vehicles (AUV)* (2012), pp. 1–5.

[LVV12] Frank L. Lewis, Draguna L. Vrabie, and Kyriakos G. Vamvoudakis. “Reinforcement Learning and Feedback Control: Using Natural Decision Methods to Design Optimal Adaptive Controllers”. In: *IEEE Control Systems* 32 (2012), pp. 76–105.

[Arg+13] Lucas M. Argentim et al. “PID, LQR and LQR-PID on a quadcopter platform”. In: *2013 International Conference on Informatics, Electronics and Vision (ICIEV)* (2013), pp. 1–6.

[BA13] Mouhacine Benosman and Gökhan M. Atinç. “Nonlinear learning-based adaptive control for electromagnetic actuators”. In: *2013 European Control Conference (ECC)* (2013).

[HA13] P. Haghi and K. Ariyur. “Adaptive feedback linearization of nonlinear MIMO systems using ES-MRAC”. In: *2013 American Control Conference* (2013), pp. 1828–1833.

[Ben14] Mouhacine Benosman. “Learning-based adaptive control for nonlinear systems”. In: *2014 European Control Conference (ECC)* (2014), pp. 920–925.

[Bel15] Richard Bellman. “Adaptive Control Processes - A Guided Tour (Reprint from 1961)”. In: *Princeton Legacy Library*. 2015.

[Dul+15] Gabriel Dulac-Arnold et al. “Reinforcement Learning in Large Discrete Action Spaces”. In: *CoRR* abs/1512.07679 (2015).

[GF15] J. Garcia and F. Fernández. “A comprehensive survey on safe reinforcement learning”. In: *J. Mach. Learn. Res.* 16 (2015), pp. 1437–1480.

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[Mni+15] V. Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518 (2015), pp. 529–533.

[Sch+15] John Schulman et al. “Trust Region Policy Optimization”. In: *ICML*. 2015.

[Sun+15] T. Sun et al. “Target following for an autonomous underwater vehicle using regularized ELM-based reinforcement learning”. In: *OCEANS 2015 - MTS/IEEE Washington* (2015), pp. 1–5.

[XB15] M. Xia and M. Benosman. “Extremum seeking-based indirect adaptive control for nonlinear systems with time-varying uncertainties”. In: *2015 European Control Conference (ECC)* (2015), pp. 2780–2785.

[Xu+15] Bing Xu et al. “Empirical Evaluation of Rectified Activations in Convolutional Network”. In: *CoRR* abs/1505.00853 (2015).

[Yan+15] R. Yang et al. “Modeling of a Complex-Shaped Underwater Vehicle for Robust Control Scheme”. In: *Journal of Intelligent and Robotic Systems* (2015).

[BKH16] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. “Layer Normalization”. In: *ArXiv* abs/1607.06450 (2016).

[Bro+16] Greg Brockman et al. “OpenAI Gym”. In: *ArXiv* abs/1606.01540 (2016).

[Fur+16] Fadri Furrer et al. “RotorS—A Modular Gazebo MAV Simulator Framework”. In: *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Ed. by Anis Koubaa. Cham: Springer International Publishing, 2016, pp. 595–625.

[Has+16] Osama Hassanein et al. “Model-based adaptive control system for autonomous underwater vehicles”. In: *Ocean Engineering* 127 (2016), pp. 58–69.

[HGS16] H. V. Hasselt, A. Guez, and D. Silver. “Deep Reinforcement Learning with Double Q-Learning”. In: *Proceedings of the AAAI conference on artificial intelligence*. 2016.

[Lil+16] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *CoRR* abs/1509.02971 (2016).

[Man+16] Musa Manhães et al. “UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation”. In: *MTS/IEEE OCEANS*. Monterey, 2016.

[McC16] Leigh McCue. “Handbook of Marine Craft Hydrodynamics and Motion Control [Bookshelf]”. In: *IEEE Control Systems Magazine* (2016).

[Osb+16] Ian Osband et al. “Deep Exploration via Bootstrapped DQN”. In: *NIPS*. 2016.

[SNK16] Pouria Sarhadi, Abolfazl Ranjbar Noei, and Alireza Khosravi. “Model reference adaptive PID control with anti-windup compensator for an autonomous underwater vehicle”. In: *Robotics Auton. Syst.* 83 (2016), pp. 87–93.

[Sch+16] Tom Schaul et al. “Prioritized Experience Replay”. In: *CoRR* abs/1511.05952 (2016).

[Sil+16] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (2016), pp. 484–489.

[SMW16] Richard S. Sutton, Ashique Rupam Mahmood, and Martha White. “An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning”. In: *J. Mach. Learn. Res.* 17 (2016), 73:1–73:29.

[Ces+17] Nicolò Cesa-Bianchi et al. “Boltzmann Exploration Done Right”. In: *ArXiv* abs/1705.10257 (2017).

[Gu+17] Shixiang Shane Gu et al. “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)* (2017), pp. 3389–3396.

[Haa+17] Tuomas Haarnoja et al. “Reinforcement Learning with Deep Energy-Based Policies”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML’17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 1352–1361.

[Sch+17] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *ArXiv* abs/1707.06347 (2017).

[Yu+17] Runsheng Yu et al. “Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle”. In: *2017 36th Chinese Control Conference (CCC)*. 2017, pp. 4958–4965. DOI: 10.23919/ChiCC.2017.8028138.

[ZS17] Shangtong Zhang and R. Sutton. “A Deeper Look at Experience Replay”. In: *arXiv preprint arXiv:1712.01275* (2017).

[Ben18] Mouhacine Benosman. “Model□based vs data□driven adaptive control: An overview”. In: *International Journal of Adaptive Control and Signal Processing* 32 (2018), pp. 753–776.

[Car18] John Carlton. *Marine propellers and propulsion*. Butterworth-Heinemann, 2018.

[Car+18] Ignacio Carlucho et al. “Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning”. In: *Robotics and Autonomous Systems* 107 (2018), pp. 71–86.

[Con+18] Edoardo Conti et al. “Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents”. In: *NeurIPS*. 2018.

[FHM18] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *ArXiv* abs/1802.09477 (2018).

[Haa+18a] Tuomas Haarnoja et al. “Learning to Walk via Deep Reinforcement Learning”. In: *CoRR* abs/1812.11103 (2018).

[Haa+18b] Tuomas Haarnoja et al. “Soft Actor-Critic Algorithms and Applications”. In: *CoRR* abs/1812.05905 (2018).

[Haa+18c] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1856–1865.

[Liu+18] Yuxuan Liu et al. “Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018), pp. 1118–1125.

[Pen+18] Xue Bin Peng et al. “DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills”. In: *ACM Trans. Graph.* 37.4 (2018). ISSN: 0730-0301.

[Pla+18] Matthias Plappert et al. “Parameter Space Noise for Exploration”. In: *6th International Conference on Learning Representations, ICLR*. 2018.

[RCB18] Don van Ravenzwaaij, Peter Cassey, and Scott D. Brown. “A simple introduction to Markov Chain Monte–Carlo sampling”. In: *Psychonomic Bulletin & Review* 25 (2018), pp. 143–154.

[SSB18] Chao Shen, Yang Shi, and Bradley J. Buckham. “Trajectory Tracking Control of an Autonomous Underwater Vehicle Using Lyapunov-Based Model Predictive Control”. In: *IEEE Transactions on Industrial Electronics* 65 (2018), pp. 5796–5805.

[Sün+18] Niko Sünderhauf et al. “The limits and potentials of deep learning for robotics”. In: *The International Journal of Robotics Research* 37 (2018), pp. 405–420.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning an introduction - Second edition*. Cambridge, Mass.: MIT Press, 2018. (Visited on 02/16/2018).

[Wan+18] C. Wang et al. “Reinforcement Learning-based Adaptive Trajectory Planning for AUVs in Under-ice Environments”. In: *MTS/IEEE OCEANS*. 2018.

[WS18] Chu-Jou Wu and Karl Sammut. “6-DoF Modelling and Control of a Remotely Operated Vehicle”. In: 2018.

[BK19] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. DOI: 10.1017/9781108380690.

[Bur+19] Yuri Burda et al. “Exploration by Random Network Distillation”. In: *ArXiv* (2019).

[KNS19] Kristoffer Borgen Knudsen, M. C. Nielsen, and I. Schjølberg. “Deep Learning for Station Keeping of AUVs”. In: *OCEANS 2019 MTS/IEEE SEATTLE* (2019), pp. 1–6.

[Kum+19] Aviral Kumar et al. “Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction”. In: *NeurIPS*. 2019.

[Pas+19] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *NeurIPS*. 2019.

[PL19] Andreea-Iulia Patachi and Florin Leon. “A Comparative Performance Study of Reinforcement Learning Algorithms for a Continuous Space Problem”. In: *2019 23rd International Conference on System Theory, Control and Computing (ICSTCC)* (2019), pp. 860–865.

[Sut19] Raik Suttner. “Extremum seeking control with an adaptive dither signal”. In: *Automatica* 101 (2019), pp. 214–222. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2018.11.055. URL: https://www.sciencedirect.com/science/article/pii/S000510981830596X.

[Tes+19] Chen Tessler et al. “Action Assembly: Sparse Imitation Learning for Text Based Games with Combinatorial Action Spaces”. In: *ArXiv* abs/1905.09700 (2019).

[Bar+20] L. Barker et al. “Scientific Challenges and Present Capabilities in Underwater Robotic Vehicle Design and Navigation for Oceanographic Exploration Under-Ice”. In: *Remote. Sens.* 12 (2020), p. 2588.

[Biz20] N. Bizon. “Global Extremum Seeking Algorithms”. In: 2020.

[Cha+20a] T. Chaffre et al. “Sim-to-Real Transfer with Incremental Environment Complexity for Reinforcement Learning of Depth-Based Robot Navigation”. In: *17th International Conference on Informatics, Automation and Robotics, ICINCO*. 2020.

[Cha+20b] Thomas Chaffre et al. “Sim-to-Real Transfer with Incremental Environment Complexity for Reinforcement Learning of Depth-Based Robot Navigation”. In: *17th International Conference on Informatics, Automation and Robotics, ICINCO 2020*. Proceedings of the 17th International Conference on Informatics, Automation and Robotics, ICINCO 2020. Virtual, Online, France, July 2020, pp. 314–323.

[Chu+20] Zhenzhong Chu et al. “Motion control of unmanned underwater vehicles via deep imitation reinforcement learning algorithm”. In: *IET Intelligent Transport Systems* 14.7 (2020), pp. 764–774. DOI: https://doi.org/10.1049/iet-its.2019.0273.

[20] “Deep Learning and System Identification”. In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 1175–1181.

[Dul+20] Gabriel Dulac-Arnold et al. “An empirical investigation of the challenges of real-world reinforcement learning”. In: *ArXiv* abs/2003.11881 (2020).

[Fed+20] W. Fedus et al. “Revisiting Fundamentals of Experience Replay”. In: *International Conference on Machine Learning*. 2020.

[Liu+20] Zhuang Liu et al. “Regularization Matters in Policy Optimization-An Empirical Study on Continuous Control”. In: *International Conference on Learning Representations*. 2020.

[Men+20] Russell Mendonca et al. “Meta-Reinforcement Learning Robust to Distributional Shift via Model Identification and Experience Relabeling”. In: *ArXiv* abs/2006.07178 (2020).

[Moh+20] Shakir Mohamed et al. “Monte Carlo Gradient Estimation in Machine Learning”. In: *J. Mach. Learn. Res.* 21 (2020), 132:1–132:62.

[Sol+20a] Yoann Sola et al. “Evaluation of a Deep-Reinforcement-Learning-based Controller for the Control of an Autonomous Underwater Vehicle”. In: *Global Oceans* (2020), pp. 1–7.

[Sol+20b] Yoann Sola et al. “Evaluation of a Deep-Reinforcement-Learning-based Controller for the Control of an Autonomous Underwater Vehicle”. In: *Global Oceans 2020: Singapore – U.S. Gulf Coast*. 2020, pp. 1–7. DOI: 10.1109/IEEECONF38699.2020.9389415.

[WEG20] Svante Wold, Kim H. Esbensen, and Paul Geladi. “Principal Component Analysis”. In: *Comprehensive Chemometrics* (2020).

[Cha+21] Thomas Chaffre et al. “Direct Adaptive Pole-Placement Controller using Deep Reinforcement Learning: Application to AUV Control”. In: *IFAC-PapersOnLine* 54.16 (2021). 13th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS, pp. 333–340.

[Ers+21] Julian Erskine et al. “Model Predictive Control for Dynamic Quadrotor Bearing Formations”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 124–130. DOI: 10.1109/ICRA48506.2021.9561304.

[Gho+21] Dibya Ghosh et al. “Why Generalization in RL is Difficult: Epistemic POMDPs and Implicit Partial Observability”. In: *NeurIPS*. 2021.

[Gon+21] Peng Gong et al. “Lyapunov-based model predictive control trajectory tracking for an autonomous underwater vehicle with external disturbances”. In: *Ocean Engineering* 232 (2021), p. 109010. ISSN: 0029-8018.

[Hay+21] Tyler L. Hayes et al. “Replay in Deep Learning: Current Approaches and Missing Biological Elements”. In: *Neural Computation* 33.11 (2021), pp. 2908–2950. DOI: 10.1162/neco_a_01433.

[Ras+21] Paria Rashidinejad et al. “Bridging Offline Reinforcement Learning and Imitation Learning: A Tale of Pessimism”. In: *NeurIPS*. 2021.

[Sil+21] D. Silver et al. “Reward is enough”. In: *Artificial Intelligence* 299 (2021), p. 103535.

[WSS21] Henglai Wei, Chao Shen, and Yang Shi. “Distributed Lyapunov-Based Model Predictive Formation Tracking Control for Autonomous Underwater Vehicles Subject to Disturbances”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51 (2021), pp. 5198–5208.

[Bak+22] Joseph Bakarji et al. “Discovering Governing Equations from Partial Measurements with Deep Delay Autoencoders”. In: 2022.

[CHA+22] Thomas CHAFFRE et al. “Learning Stochastic Adaptive Control using a Bio-Inspired Experience Replay”. In: (Mar. 2022).

[Cha+22] Thomas Chaffre et al. “Learning-based vs Model-free Adaptive Control of a MAV under Wind Gust”. In: *Informatics in Control, Automation and Robotics*. Springer, 2022, pp. 362–385.

[Koh+22] Hector Kohler et al. “PID Tuning using Cross-Entropy Deep Learning: a Lyapunov Stability Analysis”. In: *IFAC-PapersOnLine* (2022).

A Experience replay algorithms

Algorithm 1: Original ER

Initialize the value function Q
Initialize the replay buffer B
while *training is not finished* **do**
  Get the initial state S_t
  while S_t *is not terminal state* **do**
    Select an action A_t according to a ϵ-greedy policy derived from Q
    Execute the action A_t, get the reward R and the next state S_{t+1}
    Store the transition $e_t = S_t, A_t, R_t, St + 1$ into the replay buffer B
    Sample a random batch of transition D from B
    Update the value function Q with D
    $S_t \leftarrow S_{t+1}$
  end
end

Algorithm 2: CER

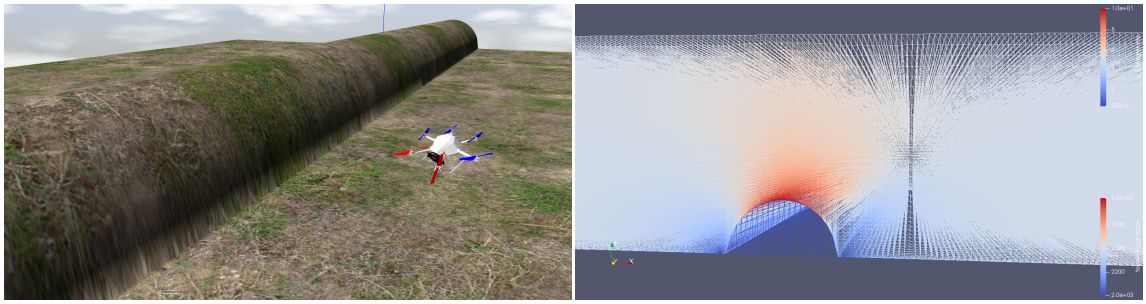
Initialize the value function Q
Initialize the replay buffer B
while *training is not finished* **do**
  Get the initial state S_t
  while S_t *is not terminal state* **do**
    Select an action A_t according to a ϵ-greedy policy derived from Q
    Execute the action A_t, get the reward R and the next state S'
    Store the transition $e_t = S_t, A_t, R_t, S_{t+1}$ into the replay buffer B
    Sample a random batch of transition D from B
    Add last transition e_t to D
    Update the value function Q with D
    $S_t \leftarrow S_{t+1}$
  end
end

Algorithm 3: BIER

Initialize the value function Q
Initialize the replay buffer B
while *training is not finished* **do**
  Get the initial state S_t
  while S *is not terminal state* **do**
    Select an action A_t according to a ϵ-greedy policy derived from Q
    Execute the action A_t, get the reward R_t and the next state S_{t+1}
    if t *is even* **then**
      Store the transition $e_t = S_t, A_t, R_t, S_{t+1}$ into the replay buffer B_1
    end
    if $R_t \geq \mathbf{E}(R)$ **then**
      Store the transition $e_t = S_t, A_t, R_t, S_{t+1}$ into the replay buffer B_2
    end
    Sample E a random temporal sequence of transition of length $D/2$ from B_1
    Sample F a random batch of $D/2$ transition from B_2
    Update the value function Q with $E + F$
    $S_t \leftarrow S_{t+1}$
  end
end

B The RotorS Simulator package

The Gazebo-based package called RotorS [Fur+16] is a high-fidelity simulation framework for MAVs that does not require any additional components to simulate high-level tasks (e.g. path-planning, collision avoidance, or vision-based problems). A complete model of the Firefly MAV is directly included in the package along with various world models and a plugin to simulate wind fields. We relied on the wind plugin provided by RotorS to generate wind fields in the environment. This plugin allows to define the wind as a 3D field sampled over a regular grid and each point specifies a wind velocity vector (in $m.s^{-1}$) which varies here between $-5m.s^{-1}$ and $+10m.s^{-1}$. As illustrated in Figure B.1, we employed the environment named *hemicyl* (see Figure B.1) and its pre-configured wind field which is composed of 6282 vertices that are represented in Figure B.1b.



(a) **(b)**

Figure B.1: *Visualization of the simulated environment in Gazebo (a) and the wind field in Paraview (b). Characteristics of the wind field can be find on the RotorS plugin page.*

The parametrization of the Firefly platform considers two reference frames: the world-fixed inertial frame $\mathcal{R}_W$ and the body reference frame $\mathcal{R}_B$ which is attached to the center of mass of the hexacopter. Coordinates in the world frame are denoted as $[x_W, y_W, z_W]^T$ while they are denoted as $[x_B, y_B, z_B]^T$ in the body frame. The pose of the hexacopter is given by its position $\zeta = [x_W, y_W, z_W]^T$ and orientation $\eta = [\phi, \theta, \psi]^T$ in the three Euler angles (respectively roll, pitch and yaw). For the sake of clarity, $\sin(\cdot)$ and $\cos(\cdot)$ are abbreviated as $s\cdot$ and $c\cdot$ in the next equation. The transformation from $\mathcal{R}_W$ to $\mathcal{R}_B$ is given by:

$$\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = \begin{bmatrix} c\theta c\psi & c\theta s\psi & -s\theta \\ s\phi s\theta c\psi - c\phi s\psi & s\phi s\theta s\psi + c\phi c\psi & s\phi c\theta \\ c\phi s\theta c\psi + s\phi s\psi & c\theta s\theta s\psi - s\phi c\psi & c\phi c\theta \end{bmatrix} \begin{bmatrix} x_W \\ y_W \\ z_W \end{bmatrix} \tag{B.1}$$

The main forces acting on the vehicle come from gravity and the thrust of the rotors. Rotors drag and air friction are neglected here to simplify the model. It is classically assumed that the hexacopter is a rigid body with a symmetrical structure, and tensions in all directions are proportional to the square of the propeller speed. The equations of motion follow as:

$$\begin{aligned} \dot{\zeta} &= v \\ \dot{v} &= -ge_3 + \mathbf{R}\left(\frac{b}{m}\sum \Omega_i^2\right) \\ \dot{\mathbf{R}} &= \mathbf{R}\hat{\omega} \\ \mathbf{I}\dot{\omega} &= -\omega \times \mathbf{I}\omega - \sum J_r(\omega \times e_3)\Omega_i \tau \end{aligned} \tag{B.2}$$

where $\mathbf{R}$ the rotation matrix from $\mathcal{R}_B$ to $\mathcal{R}_W$; ω is the skew symmetric matrix ; Ω_i is the i-th rotor speed ; $\mathbf{I}$ the body inertia ; J_r the rotor inertia ; b is the thrust factor and τ is the torque applied to the body frame due to the rotors. A classical cascade control structure is adopted [Ber+11], where the built-in low-level controller of the RotorS package is used to track a reference in roll ϕ_r, pitch θ_r and thrust $\mathcal{T}$. The yaw angle ψ is kept constant without loss of generality. Under the small-angle assumption on ϕ and θ, the guidance model reduces to the double-integrator model:

$$\begin{aligned} \dot{\zeta} &= v, \\ \dot{v} &= u = [u_x, u_y, u_z]^T, \end{aligned} \tag{B.3}$$

where the computed accelerations are converted into low-level control inputs as:

$$\mathcal{T} = m(u_z + g) \; ; \; \theta_r = \frac{m}{\mathcal{T}}(c\psi u_x + s\psi u_y) \; ; \; \phi_r = \frac{m}{\mathcal{T}}(s\psi u_x - c\psi u_y). \tag{B.4}$$

C Publications

Evaluation of a Deep-Reinforcement-Learning-based Controller for the Control of an Autonomous Underwater Vehicle

Yoann Sola*, Thomas Chaffre*†, Gilles Le Chenadec*, Karl Sammut†, Benoit Clement*†

*Lab-STICC UMR CNRS 6285 at ENSTA Bretagne, Brest, France, benoit.clement@ensta-bretagne.fr

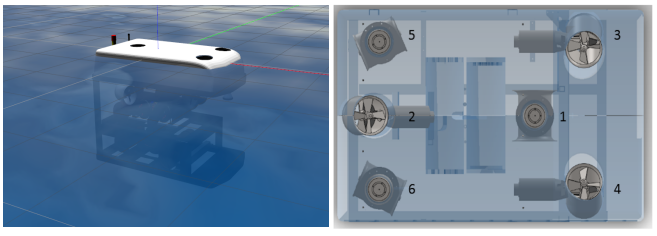
† Centre for Maritime Engineering, College of Science and Engineering, Flinders University, Adelaide, Australia, SA 5042, karl.sammut@flinders.edu.au

***Abstract*—The development of efficient controllers in underwater environments has long been a challenging topic, hindered mostly by the lack of understanding in process variations under these conditions. It is without a doubt difficult for an autonomous underwater vehicle to behave as instructed while being constantly trying to compensate for the disturbing forces that act on its body. Recently, noteworthy improvements have been made in the model-free control theory, allowing the use of Reinforcement-Learning-based controllers in terrestrial and aerial robotic contexts. In contrast, the underwater control field has been largely dominated by controllers based on the Proportional-Integral-Derivative structure. In this paper we compare a PID controller with a leading-edge Deep Reinforcement Learning algorithm, the Soft Actor-Critic. These controllers have been tested within marine robotics simulations, using a realistic simulated environment including external disturbances for a waypoint tracking mission. The results obtained reveal the superiority of PID controllers in terms of success rate and precision, but not in terms of thruster usage. Future works will include the test of these controllers with an underactuated AUV, to demonstrate the guidance abilities of the learning approach.**

***Index Terms*—AUV, control theory, deep reinforcement learning, simulation, precision, thrusters usage**

I. INTRODUCTION

The control of Autonomous Underwater Vehicles (AUVs) is a particularly challenging task. In the strongly unstructured and uncertain marine environment, AUVs must constantly try to compensate for disturbing forces such as wind, wave and current action that act on their body. The wave frequency changes with weather condition variations and will fluctuate even more since it is also influenced by the vehicle relative velocity and heading. An AUV operating in comparable environment needs to perform well on specific tasks such as path following or dynamic positioning, while being robust to unexpected events (propeller failures, seafloor rocks, other vehicles, etc.) and any type of disturbances (sea currents, sensor noize, etc.). The decision-making process of an AUV is provided by what is called a Guidance, Navigation and Control (GNC) system. The navigation function involves estimating the state of the AUV, typically its pose, its linear and angular velocities and their respective derivatives. The estimated state is then used by the guidance and the control parts. One of the most commonly used state observers is the Extended Kalman filter [1]. In this work we make the assumption that the kinematic state of the AUV is known, and we will only focus on the guidance and control aspects of the GNC system. The guidance is a high-level control subsystem, whose task is to define setpoint values for the low-level control algorithm. The type of guidance algorithm to be used is defined according to the mission we want the AUV to perform (waypoint tracking, obstacle avoidance, etc.) and may come from different areas: for example potential fields methods [2] or line-of-sight [3]. The control part is provided by low-level controllers which compute the inputs needed by the AUV's actuators (propellers and fins) in order to follow the setpoint values provided by the guidance system. Several approaches can be used in control theory. Depending on the particularities of the task, the robotic platform and the environment, one designer can use different methods: adaptive control, robust control, optimal control, sliding mode control, etc. The paper is organized as follows. In section II we provide a presentation on recent work concerning the control of AUVs and then follow this up with a discussion on the use on Reinforcement Learning in robotics. In section III we present details on our deep-reinforcement-learning-based model-free controller. Finally, simulation results and the comparison with the PID controller are assessed in section IV.



(a) Rexrov2 in the ROS-Gazebo environment [4].

(b) Thrusters layout of the Rexrov2 (top view) [5].

Figure 1: Illustration of the simulated robotic platform with its thrusters configuration and the environment used for the training and testing of both controllers.

II. RELATED WORK

A. Classic Control Approaches

The Proportional-Integral-Derivative (PID) controller is the most commonly used low-level controller for many applications. It is composed of three terms: the proportional term which is proportional to the tracking error (the difference between the controlled variable and the setpoint to reach), the derivative of the tracking error and its integral. The command law is then structured according to different strategies: one PID controller for each degree of freedom (DOF) [6] ; pairs of PID controllers, with the output of the first one becoming a setpoint value for the second one [7].

The PID controllers relies on a mathematical model of the controlled system which captures relatively well its dynamics around some operating conditions. Difficulties appear with variations of internal parameters of the system model and in the character of the disturbances. Vulnerabilities increase with variations in the process dynamics. In real-life problems, it is difficult to describe these mechanisms analytically because there are many different sources for the variations and in most cases, the underlying reasons for them are yet not fully understood.

In this study we have been focusing on approaches based on controllers which do not rely on any physical model of the system. These approaches are classified as model-free algorithms.
The recent surge in data collection and analysis has greatly supported the use of Reinforcement Learning, a machine learning technique, as an optimization method for model-free controllers. We introduce in the next subsection the main topics of this approach.

B. Deep Reinforcement Learning in Robotics

Traditional methods for the control of an autonomous vehicle usually rely on multiple model-based blocks that perform waypoint navigation algorithms, inertial and/or visual odometry and low-level control. These methods required expert knowledge and usually do not possess sufficient robustness to process variations when working in unstructured and uncertain conditions like under water. On the other hand, model-free control methods relying on Reinforcement Learning (RL), have recently shown superior performances in the control of terrestrial [8], aerial [9] and underwater robots [10][11]. The objective of RL methods is for a learning agent to find an optimal strategy behaviour (called a policy and denoted by π) from trials and errors. Those repeated interactions are in the form of the execution of an action $a_t \in A$ from state $s_t \in S$ which make the agent transit to a new state $s_{t+1} \in S$. This transition produce a feedback signal noted $r \in \mathbb{R}$ known as the reward, which quantitatively transcribes how good was this transition in regard to a given state s and with respect to a reward function $R(.)$. This type of procedure can thus be framed as a Markov Decision Process where the objective is to maximize the expected discounted total reward:

$$\min_{\theta} \sum_{t \in 1,\dots,T} \left(\left(\sum_{\tau \in 1,\dots,T} r_\tau \gamma^{\tau - t} \right) \log(\pi_\theta(a_t|s_t)) \right) \tag{1}$$

The policy $\pi(a_t|s_t)$ is generally modelled with a parameterized function with respect to θ. Two functions are exploited to improve the policy. The first one is called the Value function, denoted by $V(s)$ and represents the expected long-term reward achieved starting from a state s and then by following the policy π_θ. If this function is known, it is possible to perceive how "good" it is for the agent to be in the state s and then follow the current policy. The second function commonly used is called the Q-Value function, denoted by $Q(s,a)$ and similarly to the Value function, it measures the expected long-term reward, but this time is achieved from a state and action pair following the same policy. This quantity illustrate how good for the agent it is to take action a from state s and then follow the policy π_θ. If these functions are fully known, the given policy is optimal since, in that case, at each timestep, the agent would be aware which action to execute to transit to the next state that would maximize its total future rewards. With this purpose, many RL approaches seek to estimate these two functions in order to improve the policy. We propose now to present classical RL approaches and their application in the robotics domain.
Reinforcement-learning-based control methods can be divided in two categories: model-based and model-free approaches*. Model-based methods are strongly influenced by control theory. In this instance, a total or partial knowledge of the environment "model" x is provided which mathematically tells how it will respond to the agent's transition:

$$x_t = f(x_{t-1}, u_{t-1}) \tag{2}$$

This model is either given or learned. In the first case, during training or acting, the agent can request the model to provide a prediction of the next reward and the next state or the expected future reward or even the full distribution of future states and future rewards. If it has to be learned, a random policy is initially used to observe the trajectories. The model is then fitted using the sampled data. Standard model-based RL methods include Dynamical Programming optimization algorithms [12]. This family of methods have the strong advantage of being sample efficient. They only need a few samples to learn models since they behave linearly, at least in the local proximity. Once the model and cost function are known, we can design the optimal control without further sampling. However, even though model-based algorithms have better physical assumptions and approximations on a given task, they might succeed only for the specific task. In contrast, model-free methods uses much more samples but are able to learn directly one or several simpler quantities without any

*Here the terms model-based and model-free do not correspond to the ones used in II-A but refer to whether or not a model of the environment is given to the RL algorithm to improve the policy whereas in automation, it is the model of the controlled dynamical system that is provided.

model. Typical model-free methods are Monte Carlo Control [13], Q-Learning [14] or Actor-Critic [15] algorithms where the agent rely only on samples from the environment to change its behavior. In this study we focus on a particular type of model-free RL approach called Policy Gradient. They aim at directly modifying the parameters θ of the agent's policy to maximize its expected future discounted reward (1). In the following, we provide recent improvements made in this family of techniques.

A turning point in Policy Gradient methods arose from the paper [16] where the mechanism of Experience Replay (ER) and the use of a neural network in the Q-Learning architecture to approximate the Q-Value function were introduced. With this technique (ER), at each timestep the agent's experience $e_t = \{s_t, a_t, r_t, s_{t+1}, a_{t+1}\}$ is stored in a dataset $D = \{e_t, e_{t+1} \dots e_T\}$, pooled over every episodes into a "replay buffer". Q-Learning updates are then applied to sample experience $e \sim D$ randomly drawn from D. This arbitrary selection of samples has the benefit of reducing correlations between consecutive samples and therefore reduce the variance of the updates. When using experience replay, it is necessary to learn Off-Policy because the updated parameters are different from the ones used for sample collection. This means that a policy $\pi_{\theta'}$ has to be used to generate samples in order to optimize the policy π_θ. Thereafter, the community has started to use the term "Deep" Reinforcement Learning to refer to approaches where the latter functions are approximate using neural networks. With this procedure, V. Mnih el al. [16] demonstrated that their agent was able to outperform the best human players on numerous games of the Atari 2600 console. The principal drawback of this approach is that it can only tackle discrete action space which seems hardly applicable in a robotic context. In fact, a designer would prefer a continuous controller to allow its robots to achieve smooth and precise movements. Significant research efforts have since been made to address this issue.

In [17] in particular, a new theorem called Deterministic Policy Gradient (DPG) has been proposed to achieve Reinforcement Learning with continuous action space. The policy is now deterministic which means that, at a given state, a given agent will always choose the same action to perform, until it is updated by the Deep RL algorithm. There are no more probabilities for each action to be selected. Moreover, its parameters are updated with gradient ascent in order to maximize the total amount of future discounted rewards. In [17], the gradient of each policy parameter with respect to the objective function to maximize is ensured by the policy gradient theorem. The DPG algorithm was improved in [18], so as to define the Deep Deterministic Policy Gradient (DDPG) algorithm. As previously explained in Deep RL, the term "Deep" refers to the use of Neural Networks (NN) to replace some features of the classical DPG. The DDPG algorithm proposes an Actor-Critic architecture, where one NN is used to model the deterministic policy, called the actor, while another one is modeling the Q-Value function, called the critic. Moreover, the target values needed by the loss function of the two NNs are themselves generated by two others NN, called the target actor and the target critic with regard again to the reward function. Finally the parameters of the target actor and the target critic are updated using exponentially moving averages of the parameters of the actor and the critic NNs. This algorithm has been used widely and notably for the End-to-End navigation of robots such as in [19] for a TurtleBot platform or in [20] for a UAV.

Lastly, in [21] a policy gradient approach with remarkable sampling efficiency and exploration capabilities was proposed. By expanding the latter improvements made in policy gradients methods combined with a brand new objective function formulation, T. Haarnoja et al. [21] demonstrated that their algorithm called "Soft Actor-Critic" (SAC) outperforms the RL methods cited so far (included DDPG and PPO [22]). In this study we proposed a model-free controller based entirely on the Soft Actor-Critic for the control and the guidance of an AUV which, to our knowledge, has never been proposed. We go further by evaluating the robustness to external disturbances of our deep-reinforcement-learning-based controller and of a classical PID controller in a ROS-based realistic simulated environment (UUV Simulator package [4]) where, in addition to sensor noise and controller output noise, we simulate various high amplitude variations in sea currents to more realistically disturb the vehicle.

III. Materials

In this section, the mathematical background of the algorithm used is provided. To facilitate reader comprehension, we define the Value function as:

$$V_\pi(s) = \mathbb{E}_{a\sim\pi_\theta}[G_t|S_t = s_t] \tag{3}$$

$$\text{with } G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \text{ the discounted future reward}$$

The parameter $\gamma \in [0;1]$ is a discount factor that penalizes rewards in the far future. It is used because we consider that the future state of our agent might have higher uncertainty and that distant rewards have minor immediate benefits. The Q-Value function is defined in the same manner as:

$$Q_\pi(s,a) = \mathbb{E}_{a\sim\pi_\theta}[G_t|S_t = s_t, A_t = a_t] \tag{4}$$

A. Soft-Actor-Critic

The Policy Gradient algorithms cited so far demand a large sampling work and possess vulnerable convergence properties that require fastidious hyperparameter tuning. On the other hand, in [21] the SAC algorithm, an Off-Policy Actor-Critic-based Policy Gradient method, has been proposed to cope with these issues by, among others things, incorporating a measure of the entropy of the policy in the reward. This approach is based on the following components:

- An Actor-Critic architecture with separate policy and value function estimators.
- An Off-Policy technique that allow the reuse of previously collected samples (II-B).
- The maximization of the entropy to ensure stability and exploration.

Models with high entropy explore more and have better generalization capabilities because they are able to choose near optimal strategies during their decision making process. This means that at any timestep, for multiple actions that seem to be equally good, a policy with high entropy should assign each an equal probability to be chosen. The policy π_θ is therefore trained to maximize the expected return and the entropy at the same time:

$$J(\pi_\theta) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}} [r(s_t,a_t) + \alpha H(\pi_\theta(.|s_t))] \quad (5)$$

where $\rho_{\pi_\theta}(s_t)$ denotes the state marginal of the trajectory distribution induced by a policy $\pi_\theta(a_t|s_t)$, the term $H(.)$ is the entropy measure and α is what we call the "temperature" parameter which determine the relative importance of the entropy against the reward.

This approach aims at learning three functions with the first one being a soft Value-function $V(s_t)$ that is trained to minimize the squared residual error:

$$J_V = \mathbb{E}_{s_t\sim D}[\frac{1}{2}(V(s_t) - \mathbb{E}_{a_t\sim\pi_\theta}[Q(s_t,a_t) - \log\pi_\theta(a_t|s_t)])^2] \quad (6)$$

with D the distribution of previously sampled experiences

Secondly, a soft Q-function $Q(s_t,a_t)$ is trained to minimize the soft Bellman residual:

$$J_Q = \mathbb{E}_{s_t\sim D}\left[\frac{1}{2}\left(Q(s_t,a_t) - \hat{Q}(s_t,a_t)\right)^2\right] \quad (7)$$

$$\text{with } \hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}[V(s_{t+1})]$$

Finally, the policy parameters are learned by minimizing the expected Kullback-Leibler divergence at each step:

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim D}\left[D_{KL}\left(\pi_\theta(.|s_t)||\frac{\exp(Q(s_t,.))}{Z(s_t)}\right)\right] \quad (8)$$

where $Z(.)$ is the partition function used to normalize the distribution. Three NNs are used to approximate these functions thanks to the use of target values generated by others NNs. These functions are referred to as "soft" functions because an exponentially moving average, with a smoothing constant $\tau = 5e^{-3}$, is used to update the target value network weights (i.e. soft updates). This procedure which has already been used in other works [18][23] has proved that using a separate target value network that slowly tracks with soft updates the actual value function, improves greatly the stability of the training process. In addition, two Q-Value functions are trained independently to optimize their associate J_{Q_i} (7). The minimum of these Q-Value functions are then used in the gradients of the Value function (6) and Policy (8). As mentioned in [21], this method alternates between sampling experience from the environment with the current policy and updating the function approximators using the stochastic gradients from batches sampled from a replay buffer.

B. Training setup

Policy-Gradient-based models are particularly sensitive to the shape of the reward function. The smallest variations can lead to the convergence or not of the model, and the magnitude and the evolution of each term of the computed function must be chosen carefully. The reward function is often crafted by trial and error, since no global rules exist in the reinforcement learning theory, and the expression of the function can drastically change from one task to another. Moreover, the transfer from simulation to real world is a major concern in reinforcement learning for robotics. In this perspective, the simulated environment used during the training must be realistic enough, while not being too complex for the agent. Indeed, if the task requested is too complex, the agent might not be able to converge: for example if the state returned by the environment is not descriptive enough, or if the reward function is too constraining. A trade-off between the reward function and the complexity of the environment must therefore be found.

Since the proposed approach here only focuses on the high-level guidance and low-level control parts of a GNC system for an AUV, the navigation considerations are not tackled. Therefore we consider that the AUV has access to good estimates of its pose, its linear and angular velocities, and its tracking errors with respect to the waypoint.
The environment state $\mathbf{X}_t$ observed by the SAC algorithm at time t is given as follows :

$$\mathbf{X}_t = [\ \mathbf{x},\ \Theta,\ \mathbf{v},\ \Omega,\ \phi_e,\ \mathbf{x}_e,\ \mathbf{u}_{t-1}\ ]^T \quad (9)$$

where $\mathbf{x} = [\ x,\ y,\ z\ ]$ is the position vector of the AUV in Cartesian coordinates, $\Theta = [\ \psi,\ \theta,\ \phi\ ]$ is its orientation vector expressed with Euler angles (respectively roll, pitch and yaw), $\mathbf{v}$ is its linear velocity vector (the derivative of $\mathbf{x}$), Ω is the angular velocity vector (the derivative of Θ), ϕ_e is the error between its current yaw and the desired yaw leading directly towards the waypoint, $\mathbf{x}_e$ is the error between its current position and the desired position, corresponding to the waypoint position, and $\mathbf{u}_{t-1}$ is the vector of the inputs of the actuators computed at the previous step, at time $t-1$.

In order to perform a waypoint tracking mission, we designed the reward function r_t defined in (10). It was inspired partly by [11], where the reward function takes into account low-level variables such as linear and angular velocities and their respective references. We adapted this work for higher level of control, taking directly into account the position of

the AUV and its reference. Therefore the guidance and control parts of the GNC system of the AUV are both provided by the SAC algorithm.

$$r_t = \begin{cases} r_{toward} & \textbf{if } d_t - d_{t-1} > 0 \\ r_{backward} & \textbf{else} \\ r_{waypoint} & \textbf{if } d_t < \varepsilon \\ r_{limit} & \textbf{if } z \notin [z_{min}, z_{max}] \end{cases} \tag{10}$$

where r_t is the reward received by the agent at time t, d_t is the current relative distance between the AUV and the waypoint to reach, z_{min} and z_{max} are the authorized limits for the vertical movement z of the AUV, and ε is a strictly positive real number.

Each term appearing in (10) represents a specific feature of the global desired behavior of the AUV:

- r_{toward} is a variable reward given when the distance d_t is decreasing, which means that the AUV moves toward the waypoint. It is defined as follows :

$$r_{toward} = \lambda_1 (d_t - d_{t-1}) - \lambda_2 \|\Omega\| \tag{11}$$

  where λ_1 and λ_2 are positive weighting terms. The term weighted by λ_1 rewards large movements toward the waypoint, while the term weighted by λ_2 penalizes strong angular speeds, and promotes indirectly a softer use of the actuators.
- $r_{backward}$ is a constant negative reward given when the distance d_t is increasing, which means that the AUV moves backward the waypoint.
- $r_{waypoint}$ is a constant positive reward given to the agent when the AUV reaches the waypoint, which leads to a terminal state, ending the current episode.
- r_{limit} is a constant negative reward given to the agent when the vertical movement of the AUV exceeds the limits defined by $[z_{min}, z_{max}]$, which leads to a terminal state, ending the current episode.

We chose the value of all the parameters in order to give to the signals r_{toward} and $r_{backward}$ a magnitude of around 10, as recommended in [21]. To make the training more realistic, we randomize several parameters of the environment. Details on this procedure are given in the following section.

IV. Simulations and Results

A. Simulation setup

The method proposed in this paper has been evaluated under simulation only, by using a model of the RexROV 2 platform (see Figure 1a) included in the UUV Simulator package [4]. It is a ROS-Gazebo-based simulator, which includes packages needed for underwater robotics simulation. The RexROV 2 is a 6 DOFs cube-shaped Remotely Operated underwater Vehicle (ROV), propelled by 6 thrusters. As shown in Figure 1b, the layout of thrusters can easily change the orientation of the RexROV 2, since some of them are able to act on more than one DOF. Consequently, the DOFs of this ROV are strongly correlated, unlike other cube-shaped ROVs which generally can independently move horizontally or vertically. The control task is therefore harder in our setup than with a classical thrusters configuration. The RexROV 2 simulation model incorporates an Inertial Measurement Unit (IMU), a Doppler Velocity Log (DVL) and a pressure sensor. We will designate it as an AUV in the following.

We compared DRL-based controller to a 6 DOFs fixed-gains PID controller tuned for a precise control of the RexROV 2 and provided by the UUV Simulator. More specifically, the PID controller regulates the entire pose of the AUV: the three coordinates of the position of the AUV and the three angles of its orientation. At the beginning of each episode, we initialize the vehicle at a position $\mathbf{x} = [0, 0, -20]$ (in meters and relative to the frame attached to the Gazebo world center) with a random orientation Θ, where $\psi = \theta = 0$ and $\phi \in [0; 360]$ (in degrees and with respect to the center of mass of the vehicle body). The waypoint is placed at the beginning of each episode at a random position located 50 meters from the AUV. The sensors measurements included in our state vector incorporate added noise such as: for each $x_i \in \mathbf{X}$, $x_i = x_i + \sigma_i$ with σ_i a sample uniformly distributed over the interval $[0.05; 0.1]$ except for the past actions $\mathbf{x}_{t-1}$ where σ_i is a sample uniformly distributed over the interval $[0.01; 0.05]$ this time. We also added fluctuating sea currents to our underwater simulated environment. The current velocity $c_v \in [0; 1]$ (in $m.s^{-1}$) and angles, $(c_{ha}; c_{va}) \in [-0.5; 0.5]$ (for horizontal and vertical angles respectively in *radians*), are randomly modified every 100 timesteps during training and testing.

B. Results analysis

Simulation results are presented here for the proposed DRL-based controller, that have been trained over 1200 episodes and a standard PID controller. The training has been stopped after approximately 4 hours because convergence in terms of performance were reached (i.e. the variance of rewards was small enough, for us, to consider the policy satisfying). The evaluation consist of using both controllers for a series of 500 episodes each with the same environment parameters: AUV initial position, waypoint positions and sea currents variations. We measure their performance in term of success rate (percentage of episodes where the AUV reaches the waypoint), temporal mean and standard deviation (SD) of the distance error $d\delta$ to the ideal trajectory (the direct path from the initial position of the AUV to the waypoint location), but also in terms of thrusters usage under the form of the temporal mean of the Euclidean norm of the input vector $\mathbf{u}$ at each step (with each element of $\mathbf{u}$ being a signal between -240 and $+240$).

We can observe that the PID controller showed better performances in terms of success rate and distance error while the DRL-based controller is better in terms of thruster usage, and thus indirectly in power consumption. The difference in power consumption can be explained by the characteristics of the controllers. In fact, the PID controller incorporates the correlation between the DOFs of the RexROV 2 platform, whereas the DRL-based controller does not. This means that

Performances criterions	PID Controller	DRL Controller
Success rate (%)	96	86
Mean $d\delta$ (m)	3.81	8.67
SD of $d\delta$ (m)	3.53	5.45
Mean of $\|\|\mathbf{u}\|\|$	541.42	481.06

Table I: Evaluation results under simulation for both controllers over the same set of 500 test episodes.

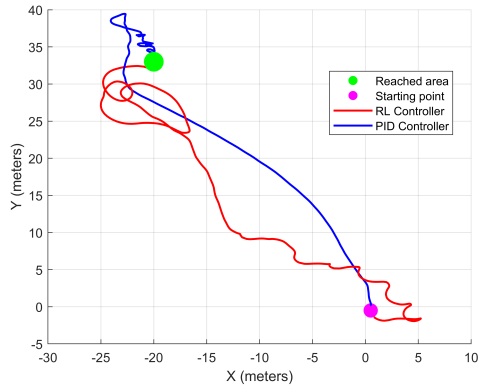
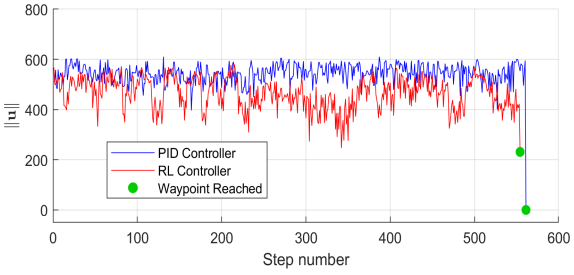
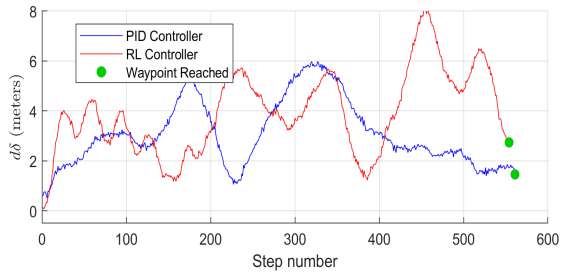


Figure 2: Trajectory followed during an episode for both controllers with $C_v = +0.096 m.s^{-}1$, $C_{ha} = -0.1324 rad$ and $C_{va} = 0.083 rad$.



(a) Euclidean norm of the input vector $\mathbf{u}$ over time steps.



(b) Distance error $d\delta$ from the ideal trajectory over time steps.

Figure 3: Performances of both controllers for the same episode characteristics as in Figure 2.

to move in a specific direction, the PID controller knows exactly which combination of thrusters to use. The DRL-based controller, on the other hand, estimates directly the inputs to the actuators and tends to use fewer thrusters than the PID controller to perform the same movement. This leads to smaller power consumption but poorer control performances than the PID controller.

V. CONCLUSIONS

In this paper we have proposed a model-free controller trained end-to-end with Deep Reinforcement Learning and compared it to a standard PID controller for the control and guidance of a 6 DOF AUV. The results provided in this study show that a PID controller is better than a DRL-based controller in terms of global success of the mission, but not in terms of actuators usage, given that the thrusters were 11.14% less solicited by the latter controller. Moreover the reward function implemented here gives satisfying results, even without taking into account the whole complexity of the task (e.g. penalizing directly strong actuators inputs, or deviations from the ideal trajectories). Proposed future directions for this research will include designing more sophisticated reward functions and comparing the two controllers in an underactuated context where the robot is not strong enough to fight the currents. In this case, the PID controller would (ineffectively) keep trying to go straight to the waypoint while the DRL-based controller should be able to discover new strategies such as letting itself be carried by the advantageous currents in order to optimize its motion toward the waypoint.

REFERENCES

[1] P. Benet, F. Novella, M. Ponchart, P. Bosser, and B. Clement, "State-of-the-art of standalone accurate auv positioning - application to high resolution bathymetric surveys," in *OCEANS 2019 - Marseille*, June 2019, pp. 1–10.

[2] D. Fu-guang, J. Peng, B. Xin-qian, and W. Hong-jian, "Auv local path planning based on virtual potential field," *IEEE International Conference Mechatronics and Automation, 2005*, vol. 4, pp. 1711–1716 Vol. 4, 2005.

[3] S. Moe, K. Y. Pettersen, T. I. Fossen, and J. T. Gravdahl, "Line-of-sight curved path following for underactuated usvs and auvs in the horizontal plane under the influence of ocean currents," *2016 24th Mediterranean Conference on Control and Automation (MED)*, pp. 38–45, 2016.

[4] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "UUV simulator: A gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey*. IEEE, sep 2016.

[5] V. Berg, "Development and commissioning of a dp system for rov sf 30k," 2012.

[6] T. I. Fossen, "Handbook of marine craft hydrodynamics and motion control: Fossen/handbook of marine craft hydrodynamics and motion control," 2011.

[7] M. M. Hammad, A. K. El-Shenawy, and M. I. E. Singaby, "Position control and stabilization of fully actuated auv using pid controller," 2016.

[8] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, pp. 2007–2014, 2018.

[9] H. Huang, Y. Yang, H. Wang, Z. Ding, H. Sari, and F. Adachi, "Deep reinforcement learning for uav navigation through massive mimo technique." 2019.

[10] Y. Sun, J. Cheng, G. Zhang, and H. Xu, "Mapless motion planning system for an autonomous underwater vehicle using policy gradient-based deep reinforcement learning," *Journal of Intelligent and Robotic Systems*, pp. 1–11, 2019.

[11] I. Carlucho, M. D. Paula, S. Wang, Y. R. Petillot, and G. G. Acosta, "Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning," *Robotics and Autonomous Systems*, vol. 107, pp. 71–86, 2018.

[12] D. P. Bertsekas, "Dynamic programming and optimal control," 1995.

[13] W. Krauth, "Introduction to monte carlo algorithms." 1998.

[14] C. J. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[15] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *NIPS*, 1999.

Learning-based vs Model-free Adaptive Control of a MAV under Wind Gust*

Thomas Chaffre[1,2], Julien Moras[3], Adrien Chan-Hon-Tong[3], Julien Marzat[3], Karl Sammut[2], Gilles Le Chenadec[1], and Benoit Clement[1,2]

[1] Lab-STICC UMR CNRS 6285, ENSTA Bretagne, Brest, France
{thomas.chaffre,gilles.le_chenadec}@ensta-bretagne.org
[2] Centre for Maritime Engineering, Flinders University, Adelaide, SA, Australia
{karl.sammut,benoit.clement}@flinders.edu.au
[3] DTIS, ONERA, Université Paris-Saclay, 91123 Palaiseau, France
{julien.moras,adrien.chan_hon_tong,julien.marzat}@onera.fr

Abstract. Navigation problems under unknown varying conditions are among the most important and well-studied problems in the control field. Classic model-based adaptive control methods can be applied only when a convenient model of the plant or environment is provided. Recent model-free adaptive control methods aim at removing this dependency by learning the physical characteristics of the plant and/or process directly from sensor feedback. Although there have been prior attempts at improving these techniques, it remains an open question as to whether it is possible to cope with real-world uncertainties in a control system that is fully based on either paradigm. We propose a conceptually simple learning-based approach composed of a full state feedback controller, tuned robustly by a deep reinforcement learning framework based on the Soft Actor-Critic algorithm. We compare it, in realistic simulations, to a model-free controller that uses the same deep reinforcement learning framework for the control of a micro aerial vehicle under wind gust. The results indicate the great potential of learning-based adaptive control methods in modern dynamical systems.

1 Introduction

Aerial vehicles are exposed to a mixture of perturbations that fluctuate vigorously because of the hazardous environment they are evolving in. They operate over a wide range of speeds and altitudes. Their dynamics are nonlinear and can also be of time-varying nature (as they fly, their mass slowly decreases due to fuel consumption and their center of gravity can greatly vary). Micro Aerial Vehicles (MAVs) are challenging in some other ways. They have to constantly compensate for the distribution of forces that act on their small airframes because of

* *This work was supported by ISblue project, Interdisciplinary graduate school for the blue planet (ANR-17-EURE-0015) and co-funded by a grant from the French government under the program "Investissements d'Avenir".*

wind and turbulence. These forces can significantly increase when weather conditions change and are also influenced by the velocity and heading of the vehicle. Because they are small, rigid, light, and move slowly, MAVs are highly sensitive to wind gusts even if mild. In addition to the latter unpredictable (external) forces, we face the difficulty of accurately modeling their (internal) dynamics under these conditions. With the development of flight controllers for hypersonic aircraft by NASA in the early 1950's [1], it was found that conventional linear feedback control strategies (e.g. fixed-gain control [2], robust control [3], sliding mode control [4] or fuzzy logic control [5]) are too limited to handle such entire regimes. It is also true for the MAVs context where for a variety of reasons it is almost impossible to manually re-tune the control parameters, these include:

- The uncertainty level of the wind speed is high and can vary more quickly than can be compensated by these types of controllers.
- The response at some operating points has to be overly conservative in order to satisfy specifications at other operating points.
- The controlled process itself varies significantly during operations (the motors' and propellers' efficiency for example).

The field of adaptive control broadly addresses these challenges by granting the control law some flexibility to modify its action based on the process variations. Despite the great advancements conducted in the theoretical area, the expansion of autonomous agents to the real-world is still limited. This is mostly due to the high dimensional problems that often appear when working under real conditions where it usually requires many degrees of freedom to describe the robot state. The lack of basic knowledge of the various natural phenomena taking place in the robot's environment, e.g. wind gust for MAVs or sea current for autonomous underwater vehicles (AUVs) makes it even more difficult to actively control it [6] with model-based adaptive methods.

For the past few years, model-free adaptive control methods have attracted significant attention. They exploit strong statistical tools that provide control systems the ability to automatically learn and improve from experience without being explicitly told how to. In particular, model-free adaptive methods based on deep reinforcement learning (DRL) have shown promising achievements. This was possible especially thanks to the use of deep neural networks to extract physical insights through sensory data (empowered by progress in data collection with high fidelity simulations, cheaper storage drives, faster computers, etc). However, as stated in [7], the extension of DRL techniques for robotic tasks raises serious questions. Compared to other application fields, robots have to interact with a dynamic environment where relevant information about the system is not always accessible or tractable over time.

In their up-to-date thorough investigations [8], G. Dulac-Arnold *et al.* presented real-world reinforcement learning challenges that are still not resolved, these include *Satisfying Environmental Constraints, High-Dimensional Continuous State and Action Spaces or Multi-Objective Reward Functions*. In addition, the notions of stability (in terms of Lyapunov stability) in the DRL framework is more deeply investigated [9] but the lack of formalism is still often highlighted

by the control community as a significant concern. The purpose of this study is to explore to what extent model-free control techniques combined with the classical theory of dynamical systems can be, at least, part of the solution to these great challenges. This is the objective of the learning-based control area [10] which aims to combine the advantages of the aforementioned paradigms (model-based and model-free) into one hybrid control scheme. Therefore, we propose a learning-based adaptive control system composed of a state-feedback control law whose parameters (i.e. gains) are tuned by the Soft Actor-Critic DRL algorithm [11] using the pole placement method. It is fairly compared, in a realistic simulation setting, with a model-free adaptive method trained end-to-end with the same DRL framework (as initially applied to robot navigation in [12]). The control objective treated is a hexacopter performing a waypoint rallying mission under unknown wind fields.

The paper is organized as follows. Section 2 gives details on the theoretical bases of the proposed control strategies and related work. In Section 3, we present background material on the simulation and control of MAVs. Section 4 is dedicated to the description of our contribution, with the application of DRL to the design of model-free and learning-based controllers. A description of the simulated training and evaluation sessions is provided in Section 5 with the obtained results. Finally, a thorough analysis of the outcomes and some suggestions to extend the work further are presented in Section 6.

2 Related work

An adaptive controller is essentially a controller that can adjust its own behavior in response to changes in the dynamics of the process and the character of the disturbances [13]. Since ordinary feedback control also seeks to reduce these undesirable effects, one needs to clearly distinguish between the two. In this paper, we will designate as *adaptive*, a control system that provides the desired system performance by changing its parameters and/or structure in order to reduce the uncertainties effect and to improve the knowledge of the desired system. As mentioned by Nikolai M. Filatov [14], the goal of adaptive systems is to switch off the adaptation as soon as the system uncertainty is reduced sufficiently and to then use the adjusted controller with a fixed structure and fixed parameters. Although it is more common in the control community to classify methods based on the nature of the model (e.g. linear vs nonlinear, continuous vs discrete), we classify here adaptive controllers based on their dependence on the mathematical model of the controlled system. This algebraic representation of the dynamics of the system can take several forms such as differential equations, state-space representations, or frequency domain representations. We can thus identify three types of adaptive control methods: totally model-based, totally model-free, and learning-based. Since the approach presented here consists in merging together some model-based controllers and model-free algorithms, in this chapter we will present the basis of the aforesaid techniques and recent applications to the problem of MAVs under unknown wind fields. Then, we will

see how reinforcement learning can be used to strengthen control laws, which will lead to the introduction of the proposed learning-based adaptive controller.

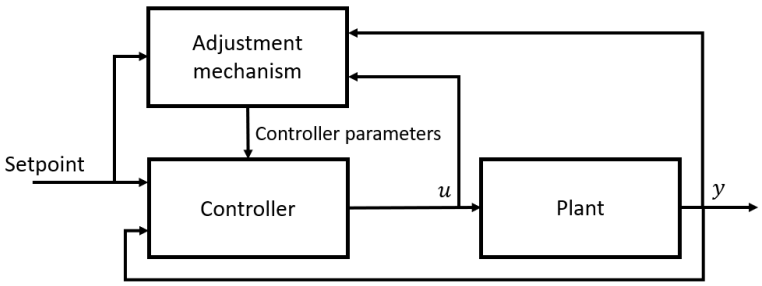


Fig. 1: General block diagram of an adaptive control system. It is composed of two loops: a normal feedback loop with the plant and the controller and a second loop with the parameter adjustment mechanism (which is often slower than the first one).

Formulation of the adaptive control problem:

Consider the system described by the following continuous-time state equations:

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t), p(t), t), \\ y(t) &= h(x(t), u(t), t), \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$ is the system state[2] ; $u \in \mathbb{R}^{n_u}$ is the control vector ; $p(t) \in \mathbb{R}^p$ is the constant or time-varying vector composed of the unknown or uncertain parameters of the model ; $y \in \mathbb{R}^m$ is the output vector of interest. The probability density of the initial values $\mathbb{P}[x(0), p(0)]$ is assumed to be known. We can define the set of outputs and control inputs available at time t as:

$$\Gamma_t = \{y(t), \ldots, y(0), u(t-1), \ldots, u(0)\}, \;\; t = 0, 1, \ldots, T-1, \;\; \Gamma_0 = \{y(0)\} \tag{2}$$

The control performance index can have the following form:

$$J(t) = \frac{1}{t} \int_0^t \bar{e}^2(\tau) d\tau, \;\; \bar{e}(t) = w(t) - y(t) \tag{3}$$

where $w(t)$ is the targeted set point. The general problem of adaptive control consists in finding the control policy $u(t) = u_t(\Gamma_t) \in \bar{\Omega}_t$ that minimizes the performance index (3) for the system described by (1) with $\bar{\Omega}_t$ the domain in the space $\mathbb{R}^{n_u}$ where the admissible control values are defined. Backward recursion

[2] In this paper, x refers to state in terms of state space representation and s in regard to the context of machine learning because although sharing the same title, these entities do not hold the same nature.

of the following stochastic continuous-time dynamic programming equations can give the optimal control of the above problem:

$$J_{t-1}(\Gamma_{t-1}) = \min_{u(t-1)\in\Omega_{t-1}} \left[\frac{1}{t}\int_0^t \bar{e}^2(\tau)d\tau|\Gamma_{t-1}\right], \tag{4}$$

$$J_t(\Gamma_t) = \min_{u(t)\in\Omega_t} \left[\frac{1}{t}\int_0^t \bar{e}^2(\tau)d\tau + J_{t+1}(\Gamma_{t+1})|\Gamma_t\right], \tag{5}$$

$$\text{for } t = t-2, t-3, \ldots, 0.$$

This optimal solution is usually impractical to derive analytically from (4) and (5) because of the dimension of the underlying spaces, even for simple cases. Near-optimal solutions can be obtained with model-based adaptive methods.

2.1 Model-based adaptive control

The design of a controller usually begins with a mathematical model of the system to be controlled. For mechanical systems of moderate dimension, it is possible to write down such a model (e.g. based on the Newtonian, Lagrangian, or Hamiltonian formalism) and eventually linearize its dynamics around a fixed point or periodic orbit. Model-based adaptive control refers to adaptive control methods that are completely reliant on this type of model. This means that the feedback control law will adjust the controller parameters online to compensate for model uncertainty. This adaptation can be performed in different manners, which gives the classification of model-based methods in two groups: direct and indirect. In direct schemes, designers attempt to estimate the control parameters. The adjustment rule tells directly how the controller's parameters should be updated. It is done without intermediate calculations involving plant parameter estimates. This is possible because in many cases, there exist measurable variables that correlate well with changes in the process dynamics. The *Gain Scheduling* [15] and *Model-Reference Adaptive Systems* [16] algorithms are examples of direct schemes. On the other hand, if the estimates of the process parameters are updated and the controller parameters are obtained from the solution of a design problem using the estimated parameters, we obtain what is called an indirect approach. This kind of controller can be seen as automation of process modeling and design, in which the process model and the control design are updated at each sampling period. The *Self-Tuning Regulators* [17] and *Dual Controllers* [18] are well-known indirect methods. Instead of further extending the theory basis of model-based control (since it is not the main focus of this study), we describe here recent applications to the control of quadrotor MAVs under wind disturbances. An output controller was proposed in [19] to cancel wind disturbances where the nonlinear dynamical mathematical model of a quadcopter was decomposed into two parts: a static MIMO transformation and few SISO channels. It was assumed that the unknown wind force acts on each channel of the quadcopter as a constant signal that has to be canceled. They proposed to first design a virtual control law for each SISO channel and

then to inverse the MIMO transformation to get the control laws for the initial considered system. Their approach showed, under simulation, that it was able to efficiently reach steady-state under unknown wind perturbations. Later on, to reduce the undesirable effects of wind on a similar quadcopter, A. Razinkova et al. [20] chose another strategy. They assumed the wind to be an unknown but uniform and time-varying force acting on the vehicle body in the X and Y-axes. Therefore, they proposed to augment a PD controller with additional terms adapting to the aforesaid forces. The resulting PD control law according to an axis i takes the form: $u_i = K_p \times e(t) + K_d \times e(t)/d_t - \gamma\tilde{i}$, where $e(t)$ is the position error and $\gamma\tilde{i}$ the adaption term. Such an adaptation converges [20] if and only if the adaptation rate γ remains positive. Later on, in [21] a $\mathscr{L}_1$ adaptive controller [22,23] (which is an advanced version of MRAC [16]) was proposed to control a quadcopter that is performing wind turbine inspection. They proved the robustness of their controller with respect to wind under simulated and real flights by comparing it to a basic Linear Quadratic Regulator (LQR). More recently, in [24] the gains of a PID controller have been tuned especially in order to handle wind gusts. They tuned this classical feedback controller in the $\mathscr{H}_2$ optimal control framework and compared it with the existing LQR-based tuning method [25]. Simulated experiments proved that the classical PID controller combined with a dedicated parameter adjustment mechanism is better at rejecting wind disturbances than the classical LQR method.

These approaches have in common the use of the *Certainty Equivalence* (CE) approach, which means that the estimation uncertainty is not taken into consideration. The wind estimates are used in the control law as if they were the real values of the unknown wind field. The CE paradigm has been used for a long time and makes it possible to generate a wide class of model-based adaptive controllers by combining different on-line parameter estimation algorithms (e.g. sensitivity methods, positivity and Lyapunov design, gradient and least-squares methods, etc) with different control laws (e.g. PID, LQG, LQR, Pole placement, etc). However, it was proved in [14] that control systems based on the CE approach are not always optimal and can be far from so. A straightforward improvement would be to apply reinforcement on the uncertainty estimates over time, in a sort of *learning process*, as a mean to optimize control performance. This goal, combined with the development of complex systems, where precise modelization is extremely difficult, has motivated the development of the model-free adaptive control field.

2.2 Model-free adaptive control

The *model-free* wording refers here to the fact that these families of controllers do not rely on any mathematical model of the controlled system. They aim at describing complex systems from observational data collected directly by embedded sensors rather than first-principles modeling. As shown in Figure 2, model-free algorithms can be seen as an optimization problem where the goal is to minimize a cost function without closed-form knowledge of the function or its gradient. A classic approach widely used in the model-free control framework

is the *Extremum-Seeking* (ES) methods [26,27]. We present now a simple ES algorithm by considering the same equations of state (1). The goal of the control is again to optimize the performance of the system described by the cost function (3). We model it as a smooth function $J(x,u) : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$. We will denote it simply by $J(u)$ because the state vector x is driven by u. In order to write convergence results we need the following assumptions:

1. There exists a smooth function $l : \mathbb{R} \to \mathbb{R}^n$ such that:

$$f(x(t), u(t), p(t), t) = 0, \text{if and only if } x(t) = l(u(t)) \tag{6}$$

2. For each $u \in \mathbb{R}$, the equilibrium $x = l(u)$ is locally exponentially stable.
3. There exists (a maximum) $u* \in \mathbb{R}$ such that:

$$\begin{aligned} (J \circ l)^{(1)}(u*) &= 0 \\ (J \circ l)^{(2)}(u*) &< 0 \end{aligned} \tag{7}$$

Based on these assumptions, one can easily design a simple extremum seeker with proven convergence bounds. One of the simplest way to maximize J is to use a gradient-based ES control law as:

$$\dot{u} = k\frac{dJ}{du}, \quad k > 0 \tag{8}$$

The convergence of (8) can be analysed with the Lyapunov function:

$$V = J(u*) - J(u) > 0, \quad \text{for } u \neq u* \tag{9}$$

The derivative of V gives:

$$\dot{V} = \frac{dJ}{du}\dot{u} = -k\left(\frac{dJ}{du}\right)^2 \leq 0 \tag{10}$$

This proves that the algorithm drives u to the invariant set $\frac{dJ}{du} = 0$, which is equivalent to $u = u*$. However, as simple as it seems, this approach requires the knowledge of the gradient of J. An ES-based model-free approach was proposed in [28] for the optimal control of a quadcopter carrying different payloads. They proved, with experimental flights, that their model-free controller is able to find the speed that maximizes the flight time (endurance) or flight distance (range) of the vehicle when transporting an unknown payload. They designed specific cost functions for each of these goals. Then, the ES algorithm was applied to estimate an optimal velocity which is used to transform the desired path into a trajectory. They proved that the ES scheme is able to find unknown, time-varying, operating points that minimize these cost functions directly from sensorial measurements and without any model of the MAV power consumption.

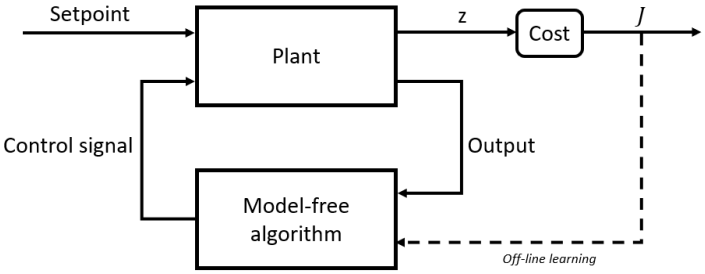


Fig. 2: General block diagram of a classical model-free controller. The control objective is to minimize a well-defined cost function J within the space of possible control laws. The off-line learning loop provides experiential data to train the controller. The vector z is composed of all the information that may factor into the cost.

Model-free control is a blossoming field where a handful of strong techniques are currently being developed and applied to minimize or maximize specific cost functions by using the system outputs as shown in Figure 2. Reinforcement learning (RL) is currently the leading technique used in this research area. It is a class of machine learning methods in which an autonomous agent in an environment has to learn how to take actions in order to maximize the notion of cumulative reward [29]. More formally, the objective of RL is for a learning agent to find an optimal strategy behavior (called a policy and denoted by π) from experimental trials and errors. Those repeated interactions are under the form of the execution of an action $a_t \in A$ from state $s_t \in S$ which makes the agent transit to a new state $s_{t+1} \in S$. This transition produces a reward signal denoted by $r(s_t, a_t) \in \mathbb{R}$, which quantitatively transcribes how well the agent is doing in the environment. The policy is a rule used by the agent to decide what actions to take and is mathematically defined as a function:

$$\pi : S \to A \tag{11}$$

This type of procedure may be framed as a Markov Decision Process [30] along a trajectory (a sequence of T actions and $T+1$ states in the environment) $\tau = (s_0, a_0, s_1, a_1, \ldots, a_{T-1}, s_T)$. Hence, the probability of a trajectory for a π policy is:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{12}$$

where $\rho_0(s_0)$ means that the starting state s_0 is randomly sampled from the distribution ρ_0 and $\pi(a_t|s_t)$ means that the action a_t is randomly sampled by the policy π in the state s_t. The overall optimization problem considered in RL is to determine the policy $\pi^\star$ which maximizes the expected return when the agent decides to take actions according to this policy:

$$\pi_\star = \arg\max J_{RL}(\pi) \tag{13}$$

where $J_{RL}(\pi)$ denotes the expected return

$$J_{RL}(\pi) = \int_{\tau} \sum_{t=0}^{T-1} r_t \cdot P(\tau|\pi) = \underset{\tau\sim\pi}{E}\left[\sum_{t=0}^{T-1} r_t\right] \tag{14}$$

Both RL and model-free paradigms are based on the same optimization formalism. We can easily see the similarity between this cost function (14) and the one from the ES algorithm (8). Note that the terms model-based and model-free can also be found in the RL theory but do not correspond to the ones used in the control field. In RL, they refer to whether or not a model of the environment (a function which predicts state transitions and rewards) is given beforehand to the RL algorithm to improve the policy whereas, in control, it is the model of the controlled dynamical system that might be provided.

Various methods exist to tackle the RL optimization problem. Among them, *Policy Gradient* techniques have enabled to use RL in real-world robotic contexts (see [31] for an extensive definition of these approaches and [32,33,34] for successful real-world robotic applications). These techniques are based on the use of a stochastic policy formally denoted by π_θ, where θ is a vector of parameters. The vector θ is estimated iteratively by gradient ascent. The expression of the gradient of $J(\pi_\theta)$ with respect to θ is given by:

$$\nabla_\theta J_{RL}(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(Q^{\pi_\theta}(s_t,a_t) - V^{\pi_\theta}(s_t))\right] \tag{15}$$

where $V^{\pi_\theta}(s_t)$ is V-value function quantifying the expected return if the trajectory starts in a state s_t taking actions upon the policy π_θ:

$$V^{\pi_\theta}(s_t) = \underset{\tau\sim\pi_\theta}{E}\left[\sum_{t=0}^{T} r_t|s_0 = s_t\right] \tag{16}$$

and $Q^{\pi_\theta}(s_t,a_t)$ is the Q-value function quantifying the expected return if the trajectory starts in a state s_t, takes an action a_t and then takes actions upon the policy π_θ:

$$Q^{\pi_\theta}(s_t,a_t) = \underset{\tau\sim\pi_\theta}{E}\left[\sum_{t=0}^{T} r_t|(s_0 = s_t, a_0 = a_t)\right] \tag{17}$$

2.3 Learning-based adaptive control

Learning-based controllers can be seen as hybrid control schemes. They are used when we only have access to an imperfect physics-based model of the system. The idea is to strengthen it with some model-free algorithm to compensate for the uncertain or the missing parts of the model. This compensation can either be done directly by *learning* the uncertain parts or indirectly by *tuning* the

controller parameters to cope with the uncertainty. To illustrate this idea, we consider the system (1) with a specific structure that can be written as:

$$\begin{aligned} \dot{x}(t) &= f_1(x(t), u(t), t) + f_2(x(t), p(t), t), \\ y(t) &= h(x(t), u(t), t) \end{aligned} \tag{18}$$

In this model, f_1 represents the known part of the plant that can be efficiently driven by a classical model-based control architecture, while f_2 represents the unknown part of the model that can be compensated by some model-free learning algorithm. The goal of learning-based controllers is to use a model-free step to optimize an unknown performance function and then use a model-based control law to guide the system's dynamics toward the optimal performance. The idea of combining both techniques is indeed attractive. One designer could take advantage of the model-based design, with its stability characteristics, and add to it the advantages of model-free learning, with its fast convergence and robustness to uncertainties. Overall, in Neural-Network (NN) learning-based control design, the idea is to write the model of the plant as a combination of a known and unknown part (i.e. the disturbances). The NNs are then used to estimate the unknown part of the model. As a result, a controller based on the known part (model-based) and the NN estimates of the uncertainties (model-free) is determined to realize some desired regulation or tracking performance. To illustrate this concept we formalize it below, in a similar spirit as the formalism from [10] where a state-space model of (1) under the Brunovsky form is considered as:

$$\begin{cases} \dot{x}_1 = x_2, \\ \dot{x}_2 = f(\cdot) + u, \end{cases} \tag{19}$$

where $f(\cdot) = f_1(x(t), u(t), t) + f_2(x(t), p(t), t)$, with f_1 known and f_2 an unknown smooth function of the state variables $x = (x_1, x_2)^T$ and u the control signal. The unknown part of the model, namely f_2, is estimated by a NN as:

$$\hat{f}_2 = \hat{W}^T S(x(t)), \tag{20}$$

where $\hat{W} = (\hat{w}_1, \ldots, \hat{w}_N)^T \in \mathbb{R}^N$ is the estimated vector of synapse weights of the neural network node and $S(x) = (s_1(x), \ldots, s_n(x))^T$ is the regressor vector, with $s_i, i = 1, \ldots, N$. Consider the reference model:

$$\begin{cases} \dot{x}_{ref_1} = x_{ref_2}, \\ \dot{x}_{ref_2} = f_{ref}(x), \end{cases} \tag{21}$$

where f_{ref} is a known nonlinear smooth function of the desired trajectories $x_{ref} = (x_{ref1}, x_{ref2})^T$. A basic learning-based controller can be defined as:

$$u = -e_1 - c_1 e_2 - \hat{W}^T S(e) + \dot{v}, \tag{22}$$

with:

$$\begin{aligned} e_1 &= x_1 - x_{ref1}, \\ e_2 &= x_2 - v, \\ v &= -c_2 e_1 + x_{ref2} \\ \dot{v} &= -c_2(-c_2 e_1 + e_2) + f_{ref}(x_{ref}), \ \ (c_1, c_2 > 0) \\ \dot{\hat{W}} &= \Gamma(S(e)e_2 - \sigma\hat{W}), \ \ (\sigma > 0 \text{ and } \Gamma^T > 0) \end{aligned} \tag{23}$$

It can be observed that u is now a function of the Brunovsky form (19) (i.e., model-based information), and the remaining part is based on the NN estimates of f_2 (model-free estimation). The unknown part is here estimated by a neural network but other strategies can be used in the learning-based scheme.

This concept of using NNs to estimate the unknown parameters (i.e. the wind field for our use case) seems appealing. In fact, as stated by the universal approximation theorem [35] for any function $f(x)$, regardless of its complexity, there exists a neural network such that for every possible input x, the value $f(x)$ is a feasible output from this network. If the external disturbances are well estimated, efficient control laws can be designed accordingly. This idea was investigated in [36,37] where a *Geometric Adaptive Controller* was proposed based on NNs for a quadcopter in wind fields. Two control laws were defined with adaptive control terms denoted as $\bar{\nabla}_i$ to mitigate the effects of the unknown disturbance. These adaptive control terms were computed with neural networks, exactly like in (22), as: $\bar{\nabla}_i = \bar{W}_i^T \varsigma(\bar{z}_i)$ with $\bar{z}_i = \bar{V}_i^T x_{nn_i}$ ($\bar{V}_i$ being the current estimate of the ideal weighting parameters and x_{nn_i} the inputs of the neural network). They proved with real experiments that the learning-based controller succeeded to complete the considered backflip maneuver followed by a stable hovering flight whereas it was not possible without the disturbance rejection terms. Another strategy was proposed in [38], where a learning-based safety-preserving cascaded QP controller (SPQC) using *Gaussian Processes* (GP) [39] has been proposed for safe trajectory tracking by a quadcopter in a cluttered environment. More precisely, the cascade controller is composed of two QP controllers: a position and an attitude level QP controller. The first one generates the desired thrust while the second makes use of it, together with the high confidence uncertainty interval obtained via GPs, to compute the desired body rotational rates. They evaluated their approach under numerical simulation whose results proved that the proposed learning-based controller is able to perform a trajectory tracking task with obstacle avoidance capacity under changing wind fields.

Compared to the approaches presented in this related work Section, we propose to only treat the parameters adjustment task. We design a learning-based controller that uses a DRL-based model-free algorithm to perform online-tuning of the parameters of a state-feedback controller. We compare this method to the model-free strategy initially applied in [12] but this time for the application of a waypoint rallying mission by a MAV under unknown wind gusts.

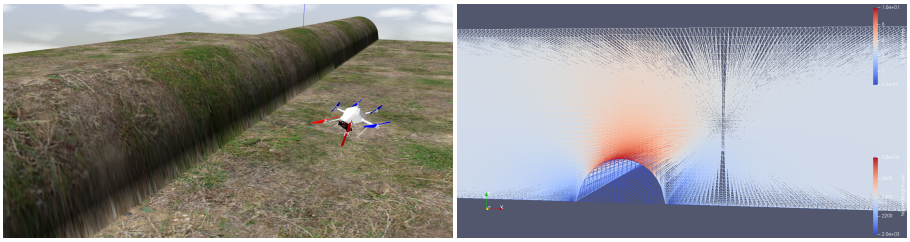
3 MAV simulation, modeling and control

This Section presents the ROS package used to simulate the aerial vehicle and wind perturbations and the corresponding model of the Firefly platform, which is the hexacopter considered in this study. The principal elements necessary to understand the control designs proposed in Section 4 are also derived.

3.1 RotorS package

Testing algorithms on physical platforms can be very time-consuming and dangerous for the robot. It can be even riskier when one designer wants to perform the training of machine learning algorithms directly on the real robot (because of their initial hazardous behavior). There exist many robotic simulation tools to reduce field testing time and make debugging easier. In this work, we relied on the ubiquitous Gazebo simulator [40] which is connected to the Robot Operating System framework (ROS [41]). ROS is an open-source meta-operating system for robots that facilitates the reuse of code.

The Gazebo-based package called RotorS [42] is a high-fidelity simulation framework for MAVs (developed by the Autonomous Systems Lab team from ETH Zurich) that does not require any additional components to simulate high-level tasks (e.g. path-planning, collision avoidance, or vision-based problems). A complete model of the Firefly MAV is directly included in the package along with various world models and a plugin to simulate wind fields. We relied on the wind plugin provided by RotorS to generate wind fields in the environment. This plugin allows to define the wind as a 3D field sampled over a regular grid and each point specifies a wind velocity vector (in $m.s^{-1}$). More specifically, we employed the environment named *hemicyl* (see Figure 3) and its pre-configured wind field, the field used is composed of 6282 vertices and the wind velocity vary here between $-5m.s^{-1}$ and $+10m.s^{-1}$.



(a) (b)

Fig. 3: Visualization of the simulated environment in Gazebo (a) and the wind field in Paraview (b). The complete details of the wind field we used can be find on the RotorS-based extended wind plugin page.

3.2 MAV model

The parametrization of the Firefly platform considers two reference frames: the world-fixed inertial frame $\mathscr{R}_W$ and the body reference frame $\mathscr{R}_B$ which is attached to the center of mass of the hexacopter. Coordinates in the world frame are denoted as $[x_W, y_W, z_W]^T$ while they are denoted as $[x_B, y_B, z_B]^T$ in the body frame. The pose of the hexacopter is given by its position $\zeta = [x_W, y_W, z_W]^T$ and orientation $\eta = [\phi, \theta, \psi]^T$ in the three Euler angles (respectively roll, pitch and yaw). For the sake of clarity, $\sin(\cdot)$ and $\cos(\cdot)$ are abbreviated as $s\cdot$ and $c\cdot$ in the next equation. The transformation from the world frame $\mathscr{R}_W$ to the body frame $\mathscr{R}_B$ is given by:

$$\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = \begin{bmatrix} c\theta c\psi & c\theta s\psi & -s\theta \\ s\phi s\theta c\psi - c\phi s\psi & s\phi s\theta s\psi + c\phi c\psi & s\phi c\theta \\ c\phi s\theta c\psi + s\phi s\psi & c\theta s\theta s\psi - s\phi c\psi & c\phi c\theta \end{bmatrix} \begin{bmatrix} x_W \\ y_W \\ z_W \end{bmatrix} \tag{24}$$

The main forces acting on the vehicle come from gravity and the thrust of the rotors. Rotors drag and air friction are neglected here to simplify the model. It is classically assumed that the hexacopter is a rigid body with a symmetrical structure, and tensions in all directions are proportional to the square of the propeller speed. The equations of motion follow as:

$$\begin{aligned} \dot{\zeta} &= v \\ \dot{v} &= -ge_3 + \mathbf{R}\left(\frac{b}{m}\sum \Omega_i^2\right) \\ \dot{\mathbf{R}} &= \mathbf{R}\hat{\omega} \\ \mathbf{I}\dot{\omega} &= -\omega \times \mathbf{I}\omega - \sum J_r(\omega \times e_3)\Omega_i \tau \end{aligned} \tag{25}$$

where $\mathbf{R}$ the rotation matrix from $\mathscr{R}_B$ to $\mathscr{R}_W$; ω is the skew symmetric matrix ; Ω_i is the i-th rotor speed ; $\mathbf{I}$ the body inertia ; J_r the rotor inertia ; b is the thrust factor and τ is the torque applied to the body frame due to the rotors. A classical cascade control structure is adopted [43], where the built-in low-level controller of the RotorS package is used to track a reference in roll ϕ_r, pitch θ_r and thrust $\mathscr{T}$. The yaw angle ψ is kept constant without loss of generality. Under the small-angle assumption on ϕ and θ, the guidance model reduces to the double-integrator model:

$$\begin{aligned} \dot{\zeta} &= v \\ \dot{v} &= u = [u_x, u_y, u_z]^T \end{aligned} \tag{26}$$

where the computed accelerations are converted into low-level control inputs as:

$$\begin{aligned} \mathscr{T} &= m(u_z + g) \\ \theta_\mathrm{r} &= \frac{m}{\mathscr{T}}(c\psi u_x + s\psi u_y) \\ \phi_\mathrm{r} &= \frac{m}{\mathscr{T}}(s\psi u_x - c\psi u_y) \end{aligned} \tag{27}$$

3.3 Controller parametrization

For future use by the learning-based control approach (Section 4.4), a specific parametrization is adopted to control the system (26). The control objective is to stabilize the robot at a waypoint ζ_{ref} with a velocity $v = 0$. Defining $e = \zeta - \zeta_{\text{ref}}$ and taking into account unmodeled disturbances (e.g. wind) by considering an additional steady-state error variable $z = \int_0^t e(\tau)d\tau$, the augmented model with state vector $X = [z, e, v]$ becomes:

$$\begin{bmatrix} \dot{z} \\ \dot{e} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} z \\ e \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \tag{28}$$

The corresponding state-feedback controller, equivalent to a PID, is such that:

$$u = -k_i z - k_p e - k_d v \tag{29}$$

The poles of the closed-loop system are solution to the following equation:

$$\lambda^3 + \lambda^2 k_d + \lambda k_p + k_i = 0 \tag{30}$$

The controller has been re-parametrized using pole placement so as to guarantee convergence to steady-state without oscillations. This way, the action space for learning purposes is limited to desired solutions in the real part of the pole map, which prevents from sampling unnecessary solutions in the space of the control gains. The desired constants $\tau_1 > 0, \tau_2 > 0, \tau_3 > 0$ are then defined as:

$$\lambda_1 = \frac{-1}{\tau_1} \, ; \, \lambda_2 = \frac{-1}{\tau_2} \, ; \, \lambda_3 = -\frac{1}{\tau_3} \tag{31}$$

Since each one is solution to (30), it follows that:

$$\begin{bmatrix} 1 & \frac{-1}{\tau_1} & \frac{1}{\tau_1^2} \\ 1 & \frac{-1}{\tau_2} & \frac{1}{\tau_2^2} \\ 1 & \frac{-1}{\tau_3} & \frac{1}{\tau_3^2} \end{bmatrix} \begin{bmatrix} k_i \\ k_p \\ k_d \end{bmatrix} = \begin{bmatrix} \frac{1}{\tau_1^3} \\ \frac{1}{\tau_2^3} \\ \frac{1}{\tau_3^3} \end{bmatrix} \Leftrightarrow MK^T = N \tag{32}$$

Finally, the gains of the controller (29) are obtained as $K^T = M^{-1}N$:

$$\begin{aligned} k_i &= \frac{1}{\tau_1 \tau_2 \tau_3} \\ k_p &= \frac{\tau_1 + \tau_2 + \tau_3}{\tau_1 \tau_2 \tau_3} \\ k_d &= \frac{\tau_1 \tau_2 + \tau_1 \tau_3 + \tau_2 \tau_3}{\tau_1 \tau_2 \tau_3} \end{aligned} \tag{33}$$

4 Methodology

In this Section, we present how to build a model-free and learning-based controller with a DRL framework composed of two components: the Soft Actor-Critic algorithm and the Experience Replay technique. Later, we present how to apply it to the model-free and learning-based control schemes.

4.1 Soft Actor-Critic

In this study, we used the same policy gradient algorithm as in our previous paper [12] to either maximize the RL-based cost function of a model-free controller (see Section 2.2) or to estimate the controller parameters of a learning-based controller (as described in Section 2.3). The *Soft Actor-Critic* (SAC) algorithm proposed by T. Haarnoja et al. in [11] is based on three concepts which we formally define thereafter.

First, the SAC algorithm uses an actor-critic architecture which concurrently learns both State value and Q-value functions and the policy $\pi_\theta(a_t|s_t)$ as well. During the training stage, the critic part updates parameters of the *State* or *Q*-value functions while the actor network updates the policy parameters in the direction suggested by the critic. Note that a recent version of the SAC algorithm [44] allows to only estimate the Q-value function, while we used in this work the original version of the algorithm [11] where the State value function is also estimated to help stabilizing the overall training process. Precisely, our implementation of the SAC algorithm aims at iteratively learning three functions modelized by three NNs:

- a policy function $\pi_\theta(a_t|s_t)$ parameterized by the NN weights θ,
- a soft Q-value function $Q_w(s_t, a_t)$ parameterized by the NN weights w,
- and a soft State value function $V_\Psi(s_t)$ parameterized by the NN weights Ψ.

Beyond their modelization, these soft Q-value and State value functions are induced by the policy π_θ and defined as follows:

$$Q_w^{\pi_\theta}(s_t, a_t) = r_t(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim\rho_\pi(s)}[V_\Psi^{\pi_\theta}(s_{t+1})] \tag{34}$$

$$V_\Psi^{\pi_\theta}(s_t) = \mathbb{E}_{a_t\sim\pi_\theta}[Q_w^{\pi_\theta}(s_t, a_t) - \alpha \log \pi_\theta(a_t|s_t)] \tag{35}$$

The terms $\rho_\pi(s)$ and $\rho_\pi(s, a)$ denote the state and the state-action marginals of the state distribution induced by the policy $\pi_\theta(a|s)$. The NNs parameters w and Ψ of the Q-value and State value functions respectively, are the solutions minimizing the soft Bellman residual $J_Q(w)$ and the mean squared error $J_V(\Psi)$:

$$J_Q(w) = \mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}\left(Q_w^{\pi_\theta}(s_t, a_t) - \Big(r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim\rho_\pi(s)}[V_{\bar{\Psi}}^{\pi_\theta}(s_{t+1})]\Big)\right)^2\right] \tag{36}$$

$$J_V(\Psi) = \mathbb{E}_{s_t \sim D}\left[\frac{1}{2}(V_\Psi(s_t) - \mathbb{E}[Q_w^{\pi_\theta}(s_t, a_t) - \log \pi_\theta(a_t, s_t)])^2\right] \tag{37}$$

with $\bar{\Psi}$ the target value function. The NN parameters θ of the policy are updated in order to minimize the expected Kullback-Leibler divergence:

$$J_\pi(\theta) = \mathbb{E}_{s_t \sim D}\left[D_{KL}\left(\pi_\theta(\cdot|s_t)\middle|\middle|\frac{\exp(Q_w^{\pi_\theta}(s_t, \cdot))}{Z_w^{\pi_\theta}(s_t)}\right)\right] \tag{38}$$

with $Z_w^{\pi_\theta}$ a partition function used to normalize the distribution. The $(\cdot) \sim D$ means that the expected values are computed using the pair (s_t, a_t) and (s_t) sampled from a replay buffer D (see Section 4.2) in which the agent experience $e_t = (s_t, a_t, r_t, s_{t+1})$ is stored at each time-step. Moreover, the authors use an exponentially moving average, with a smoothing constant $\tau = 5e^{-3}$, to update the target value network weights w (the parameters Ψ are also update in the same way since they are directly related to w). Thus these weights are constrained to change slowly from one iteration to another. The name *Soft Actor-Critic* is based on this soft update procedure. Secondly, the SAC algorithm optimizes a stochastic policy in an off-policy manner [45]. This means that the policy used to explore the environment is different from the one that is being evaluated and improved. Finally, the policy is trained to maximize simultaneously the expected return and its entropy, which is an amount of informative randomness:

$$J_{SAC}(\pi) = \sum_{t=1}^{T} \mathbb{E}_{(s_t, a_t) \sim \rho_\pi}[r(s_t, a_t) + \alpha H(\pi_\theta(.|s_t))] \tag{39}$$

where:

$$H(\pi_\theta(.|s)) = -\sum_{a \in A} \pi_\theta(a) \log \pi_\theta(a|s) \tag{40}$$

The term $H(\pi_\theta)$ is the entropy measure of policy π_θ and α is a fixed temperature parameter that determines the relative importance of the entropy term (an automatic adjustment mechanism for α was proposed in [44] by considering the constrained optimization problem of maximizing the expected return while satisfying a minimum entropy constraint but was not used in our implementation). The authors state that entropy maximization leads to policies that have better exploration capabilities. Note that this RL problem (39) is different from the initial RL problem (13). The resulting Q-value and V-value functions will therefore also include this entropy term. Two additional Q-functions are used to reduce positive bias in the policy improvement step, which is known to degrade performance of value based methods [46,47]. They are trained independently and the minimum of the two Q-functions is used as proposed in [47].

The SAC algorithm is part of the family of deep reinforcement learning approaches which try to exploit the strong representation capabilities offered by NNs to represent the Q-value, State-value and policy functions. The NNs structure of our SAC implementation is shown in Figure 4.

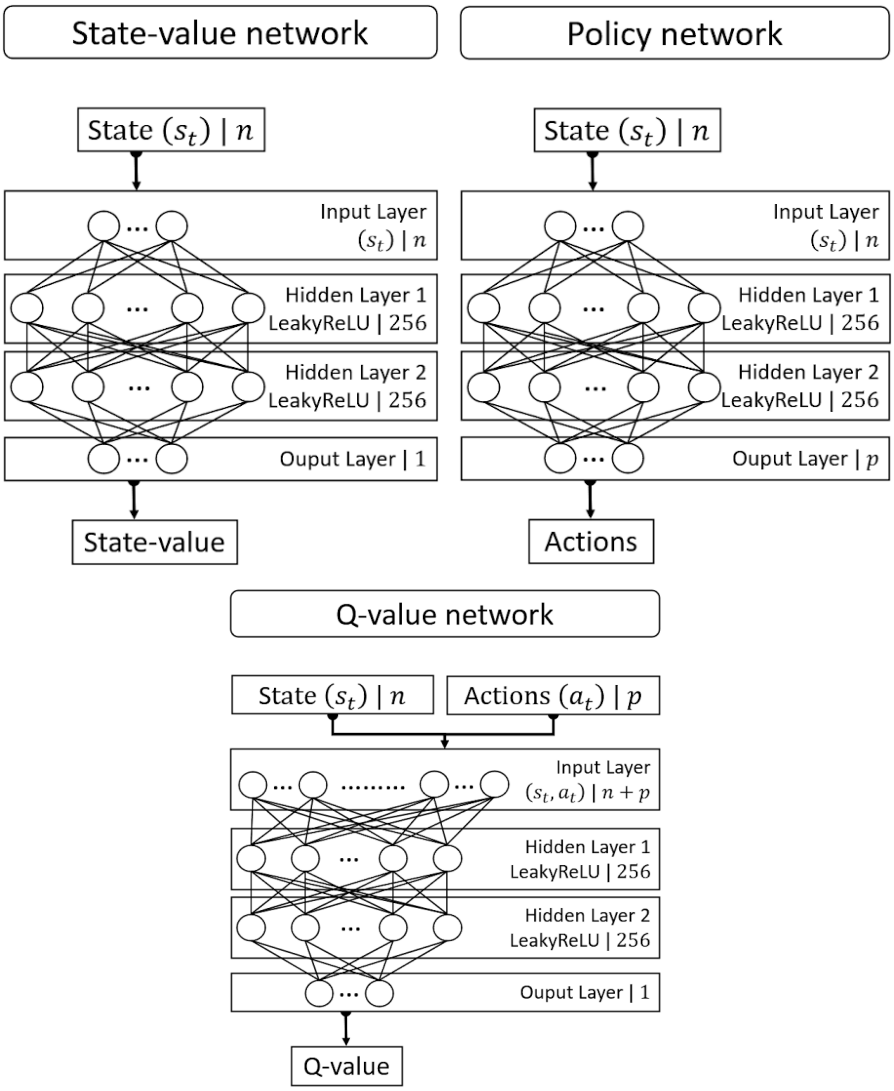


Fig. 4: The neural networks structure for our implementation of the SAC. The networks are composed of dense layers only and of two hidden layers for each network. Each layer is a fully-connected layer represented by its type, output size and activation function. The networks uses the same optimizer (Adam [48]), activation function (Leaky Relu [49]) and learning rate $l_r = 3e^{-4}$. Other parameters are a discount factor $\gamma = 0.99$ for (34), a number of 256 hidden units per hidden layer and one gradient update is performed at each time-step.

4.2 Experience Replay

To improve its performance, one needs to learn from its past actions. In [52], L. Lin showed that past experience of the learning agent should and could be used in an effective way by using what he called *Experience Replay* (ER). An *experience* is the result of a transition to a new state which can be framed as a quadruplet $e_t = (s_t, a_t, r_t, s_{t+1})$. The ER technique consists in storing at each time-step the experience e_t of the agent in a data-set (a replay buffer) $D = e_1, \ldots, e_n$ of fixed and substantial size. During training, the RL algorithm updates are then applied to mini-batches of experiences randomly pooled over the stored samples to reinforce the functions estimates.

This intuitive concept can be refined by distinct means. It seems natural that when learning a new task, some experience might be more valuable than the others and should be used more often in the update process. This relevance is in general not directly accessible, therefore a specific criterion has to be chosen to quantify it. In [53], T. Schaul proposed the *Prioritized Experience Replay* (PER) technique which consists in sampling more often from the replay buffer the transitions with high expected learning progress, as measured by the magnitude of their temporal-difference (TD) error. By doing so, they were able to obtain state-of-the-art performance on the Atari 2600 benchmark suite.

Nevertheless, depending on the task, PER can be very computationally expensive. Another parameter that has been considered is the replay capacity that is the total number of transitions possibly stored in the replay buffer. In [54], S. Zhang showed that both large and small size replay buffers can significantly damage the performance. They proposed Combined Experience Replay (CER) to cope with this problem. It consists of adding the latest transition to the mini-batch pooled over the replay buffer which only requires $O(1)$ extra computation compares to PER to reduce the negative effect of the buffer size. The difference with PER is that by using CER, the latest transitions will undoubtedly be sampled. We decided to use the CER technique and to keep a mini-batch composed of randomly sample experiences e_{rd_n} and the last one as:

$$\text{mini-batch} = [e_{rd\_1} \ ; \ e_{rd\_2} \ ; \ \ldots \ ; \ e_{rd\_n-1} \ ; \ e_{-1}] \tag{41}$$

with $n = 256$, the size of our mini-batch. We kept a replay capacity of 10^6 as recommended in the original paper of the SAC [11].

4.3 Application to the model-free controller

We will now present the use of the previous deep reinforcement learning framework to build a model-free controller. The methodology follows the same line of thought as our previous work [12]. The objective of the learning agent is to build a predictive model, using the SAC algorithm, that directly maps the actions (27) that control the MAV from the current state:

$$\pi_\theta : \ s_t \rightarrow [\ \phi_{\mathrm{r}} \ ; \ \theta_{\mathrm{r}} \ ; \ \mathcal{T} \ ] \tag{42}$$

where (ϕ_{r},θ_{r})$\in [-\frac{\pi}{6};+\frac{\pi}{6}]$ and $\mathcal{T} \in [m \times g; m \times (g+3.0)]$ (with $m = 1.544$Kg, $g = 9.81\mathrm{m.s}^{-1}$). In order to achieve the target rallying mission, we need to provide relevant data to the agent. Therefore, we defined the following vector o_t as the observation at the time-step t of the environment:

$$o_t = [\ a_{t-1}\ ;\ \phi\ ;\ \theta\ ;\ \psi\ ;\ v_x\ ;\ v_y\ ;\ v_z\ ;\ \omega_\phi\ ;\ \omega_\theta\ ;\ \omega_\psi\ ;\ \zeta_t\ ;\ e_t\ ;\ d_t\ ] \tag{43}$$

where a_{t-1} are the last actions performed ; $[\phi;\theta;\psi]$ are the Euler angles of the robot (roll, pitch and yaw) ; $[v_x;v_y;v_z]$ and $[\omega_\phi;\omega_\theta;\omega_\psi]$ are its linear and angular velocities respectively in $\mathscr{R}_W$ and $\mathscr{R}_B$; ζ_t represents its position in $\mathscr{R}_W$ (obtained from its embedded odometry sensor), $e_t = [e_x;e_y;e_z]$ are the current errors on the set-points in terms of Euclidean distance and d_t is the current Euclidean distance between the hexacopter and the target in $\mathscr{R}_W$. All the variables involved are assumed to be measured, their estimation is out of the scope of this work. We denote the target position by $\Lambda = [\Lambda_x;\Lambda_y;\Lambda_z]^T \in \mathscr{R}_W$. The dimension of this observation vector is 19 and it has been standardized to have zero mean and a variance of 1. In order to provide a higher time horizon to the agent, we constructed the state vector out of the current and past two observations vectors $[o_t;o_{t-1};o_{t-2}]$. For the purpose of providing the agent a sense of "velocity" in the evolution of the state, we consider the two by two difference of these vectors such as $vel_t = (o_t - o_{t-1})$ and $vel_{t-1} = (o_{t-1} - o_{t-2})$. We went even further and tried to add a sense of "acceleration" in the evolution of the state by including the difference between the latter vectors $acc_t = (vel_t - vel_{t-1})$. The resulting state vector is therefore defined as:

$$s_t = [\ o_t\ ;\ o_{t-1}\ ;\ o_{t-2}\ ;\ vel_t\ ;\ vel_{t-1}\ ;\ acc_t\ ] \tag{44}$$

The dimension of this state vector is 114. The reward function r_t has been designed in order to teach the agent how to complete the mission of target rallying:

$$r_t = \begin{cases} r_{receded} & \textbf{if} \quad d_{rate} \leqslant 0 \\ r_{forward} & \textbf{if} \quad d_{rate} > 0 \\ r_{reached} & \textbf{if} \quad d_t \leqslant d_{reached} \\ r_{failed} & \textbf{if} \quad z_w \notin [0.25;20] \end{cases} \tag{45}$$

where r_t is the reward of the agent at time t ; d_{rate} is the distance rate to the target performed between the last two steps such as $d_{rate} = d_t - d_{t-1}$; $d_{reached}$ is the limit value beneath which we consider the target to be reached ; $z_w \in \mathscr{R}_W$ is the MAV altitude; both $r_{reached}$ and r_{failed} are terminal rewards determined at the end of the ongoing episode. Each of these terms represent the specific features of the desired behavior of the MAV:

- $r_{receded}$ is a constant negative reward equal to -20 that is sent to the agent whenever the robot is getting away from the target (or staying immobile).
- $r_{forward}$ is a positive reward sent when the relative distance to the target is decreasing. It is defined as:

$$r_{forward} = C_1 \times e^{\left(-\left[(\frac{d_t}{1+d_{rate}})\times\frac{1}{C_2}\right]^2\right)}$$

With this design, we encourage the robot to move toward the target as fast as possible. We chose the value of the constants C_1 in order to scale the positive reward signal. This is particularly important because as mentioned in [11], the SAC algorithm is particularly sensitive to the scaling of the reward signal which is the magnitude of the reward value. We followed the recommendation prescribed in [11] and chose to set $C_1 = 20$ in order to obtain a positive reward scale of 20 (which we found in practice to be the reward scale that gave us the best performance for this problem). The constant C_2 represents how sparse is $r_{forward}$ based on the distance to the target. We chose the value $C_2 = 20$ empirically. Therefore, the positive reward is equal to:

$$r_{forward} = 20 \times e^{\left(-\left[\left(\frac{d_t}{1+d_{rate}}\right)\times\frac{1}{20}\right]^2\right)} \tag{46}$$

- A constant positive reward is sent to the agent when it succeeded to complete the mission, meaning $d_t \leq d_{reached}$. This generates $r_{reached} = +1000$.
- If the MAV altitude z_w exceed a predefined threshold, the constant negative reward $r_{failed} = -550$ is generated.

We define as *sampling rate*, the rate at which a state is sampled from the environment after the execution of the actions. A good practice consists in synchronizing it with the slowest sensor of the system which ensure no potential loss of information in the state vector and the fastest sampling rate. We synchronized it with the embedded odometry sensor which led to a *sampling rate* of about 20Hz.

4.4 Application to the learning-based controller

As stated in Section 2.3, learning-based methods consist in exploiting standard model-based control architectures that are either tuned or redesigned by a model-free algorithm in order to compensate for the unknown part of the model. The MAV is subject to an additive but unknown wind perturbation. Therefore, we combined the PID controller (29) under parametrization (33) within the DRL framework (Section 4.1) to iteratively auto-tune the feedback gains parameters (i.e. the constants τ_1, τ_2 and τ_3), with the ambition of optimizing online a desired performance cost function. For behavior stability purpose, instead of directly trying to estimate the values of τ_i, the SAC algorithm estimates at each time-step small increments ∇_i that are added to the τ_i in order to cope with the wind disturbances. By doing so, we avoid jumping from a configuration of gains to a totally different one, which usually give rise to undesired oscillations. The gains (33) are then computed using $\tau_i = \tau_i + \nabla_i$ with $\nabla_i \in [-0.01; +0.01]$ (while making sure that $3 \geq \tau_1 \geq \tau_2 \geq \tau_3 \geq 5 \times 10^{-3}$) and the resulting control law (29) is finally applied. The τ_i parameters are initialized to the nominal configuration $\tau_1 = 1, \tau_2 = 2.5, \tau_3 = 0.875$. The objective of the learning agent is now to build a predictive model that directly maps ∇_i from the current state to adjust the gains starting from this inital configuration:

$$\begin{aligned} \pi_\theta : \ s_t \rightarrow [\ & \nabla_{\tau_{1_{roll}}} \ ; \ \nabla_{\tau_{2_{roll}}} \ ; \ \nabla_{\tau_{3_{roll}}} \ ; \ \nabla_{\tau_{1_{pitch}}} \ ; \\ & \nabla_{\tau_{2_{pitch}}} \ ; \ \nabla_{\tau_{3_{pitch}}} \ ; \ \nabla_{\tau_{1_{thrust}}} \ ; \ \nabla_{\tau_{2_{thrust}}} \ ; \ \nabla_{\tau_{3_{thrust}}} \ ] \end{aligned} \tag{47}$$

We used a slightly different observation vector from the model-free scheme (43). Indeed, we propose to add the resulting parameters and the PID controller outputs (29) in the observation vector:

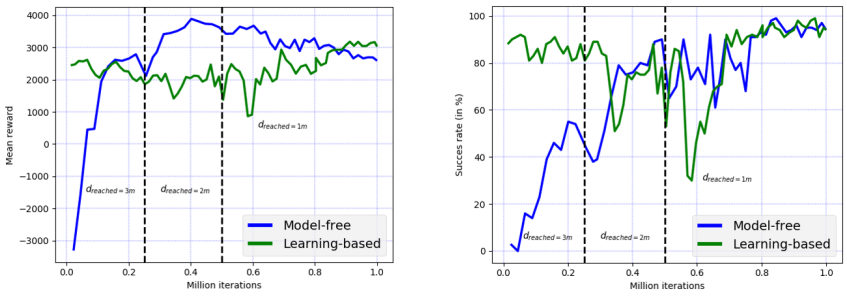
$$\begin{aligned} o_t = [\ & a_{t-1}\ ;\ \tau_{rpt}\ ;\ pid_{rpt}\ ;\ k_{p_{roll}}\ ;\ k_{i_{roll}}\ ;\ k_{d_{roll}}\ k_{p_{pitch}}\ ;\ k_{i_{pitch}}\ ; \\ & k_{d_{pitch}}\ ;\ k_{p_{thrust}}\ ;\ k_{i_{thrust}}\ ;\ k_{d_{thrust}}\ ;\ \phi\ ;\ \theta\ ;\ \psi\ ;\ v_x\ ;\ v_y\ ; \\ & v_z\ ;\ \omega_\phi\ ;\ \omega_\theta\ ;\ \omega_\psi\ ;\ \zeta_t\ ;\ e_t\ ;\ d_t\ ] \end{aligned} \tag{48}$$

where $dim(\tau_{rpt}) = 9$ and $pid_{rpt} = [\phi_{\mathrm{r}};\ \theta_{\mathrm{r}}\ ;\ \mathcal{T}]$. The dimension of this observation vector is 37. We constructed the state vector out of this observation vector as earlier (44), resulting to a state vector of dimension 222. We used the exact same reward function and parameters as described earlier (45) to train this controller.

5 Results

A training episode is defined as follows: at the beginning of the episode, the MAV is set at the center of the environment at an altitude of 3 meters with roll, pitch and yaw angles equal to 0. A target is then initialized at a fixed and uniformly random position $\Lambda = [\Lambda_x; \Lambda_y; \Lambda_z]^T$ with $[\Lambda_x; \Lambda_y]^T \in [-20; -5[\cup]5; 20]$ and $\Lambda_z \in [2; 20]$. The mission then begins and is considered as a success if the relative distance to the target is inferior to a predefined threshold $d_{reached}$ and as a failure if an error signal is generated, both cases ending the episode. Otherwise, the episode is ended if the number of iteration steps reach the maximum value allowed per episode that is set at 300. The training for both controllers consisted in performing 1 000 000 iterations in a environment with a varying wind field (as described in Section 3.1). To help the agent, $d_{reached}$ is reduced during training as follows: at first $d_{reached} = 3$m, from iteration 250 000th we set $d_{reached} = 2$m and from iteration 500 000th we set $d_{reached} = 1$m. The PyTorch framework [50] was used to carry out the numerical experiments, along with the CUDA toolkit [51] and an RTX 2060 GPU card, allowing us to perform the training of one controller in approximately 10 hours. It can be seen in Figure 5 that during training, both the learning-based (LB) and model-free (MF) controllers were able to reach a high success rate under unknown wind gust disturbances, which shows the applicability of the SAC DRL procedure for this type of aerial navigation problems. The LB strategy presents a much higher convergence speed to a significant success rate than the MF strategy, which shows the great potential of combining model-based classical controllers with learning procedures. This can be explained by the model-based part of the LB controller which allows it to choose relatively good actions despite being at the early stage of the training session. Therefore, from the beginning of the training the LB controller is able to explore a much higher part of the reward space than the one of the MF controller. We believe this significantly helps the DRL algorithm to find better overall strategies. The evaluation consisted in performing the target rallying mission in areas of the same environment that had never been explored by neither controllers during training (i.e. the wind field in these areas were

totally unknown to the neural networks) with $d_{reached} = 1$m. The evaluation is composed of a total of 500 episodes, different from each other in terms of target position and with a max step size per episode of 1000. The targets during evaluation were uniformly distributed in the space defined by $\Lambda = [\Lambda_x; \Lambda_y; \Lambda_z]^T$ with $[\Lambda_x; \Lambda_y]^T \in [-50; -20[\cup]20; 50]$ and $\Lambda_z \in [2; 20]$. The same set of evaluation episodes was used for each controller. We also evaluated a fixed control strategy which consisted in a PID controller with the fixed nominal poles configuration (4.4). The outcomes of this evaluation are provided in Table 1. We can see that the LB controller achieves a better performance for each metric. Noted that it requires, on average, less actions to achieve the task with the LB controller despite sharing the same *sampling rate*. The mean reward per step of the LB controller is more than 2 times higher than the MF one. On the other hand, the fixed control strategy is showing a much lower success rate, close to 50%. We observed that with this strategy, failures mostly consisted in cases where the MAV is close to the steady-state but is being deviated by the wind gust. The vehicle is then not able to recover with fixed-pole parametrization. It shows the benefits of adaptive control methods when facing unknown disturbances.



(a) Mean reward (b) Success rate

Fig. 5: Training curves showing the mean reward and success rate computed per 100 episodes over a moving window of 100 episodes. Note that with SAC, the policy is trained to maximize also the entropy, therefore the mean action does not always correspond to the optimal action for the maximum return objective.

Controller type	Mean step number	Mean total reward	Mean reward per step	Success rate	Positive reward rate
Fixed Poles PID	488	710.797	1.454	50.6%	61.197%
Model-free	357	2238.970	6.256	74.2%	86.056%
Learning-based	**281**	**4034.434**	**14.346**	**91.6%**	**89.2%**

Table 1: Evaluation results.

6 Conclusions

We present a novel application of the SAC procedure for learning both model-free and learning-based adaptive controllers applied to the autonomous navigation of a MAV under wind gust conditions. These two strategies have been trained and compared in the same ROS-Gazebo reference simulation. Both strategies were able to reach high levels of performances under unknown uncertainty conditions, but the learning-based scheme, with a judicious parametrization of its model-based control part, exhibited a much faster convergence rate. Our results suggest that learning-based adaptive methods can be much more efficient than model-free ones and allow a better stability analysis in the DRL scheme.

Acknowledgements The authors thank Dr. Estelle Chauveau from Naval Group for the help provided. This work was supported by SENI, the research laboratory between Naval Group and ENSTA Bretagne.

References

1. Z. T. Dydek and A. M. Annaswamy and E. Lavretsky: *Adaptive Control and the NASA X-15-3 Flight Revisited.* In: IEEE Control Systems Magazine, Vol. 30, nb. 3, pp. 32-48, 2010.
2. William J. Terrell: *Some fundamental control Theory vol. 2: Feedback linearization of single input nonlinear systems.* In: American Mathematical Monthly, 1999.
3. John C. Doyle: *Robust and optimal control.* In: IEEE Conference on Decision and Control, 1995.
4. Vadim I. Utkin: *Sliding Modes in Control and Optimization.* In: Communication and Control Engineering Series, 1992.
5. R. E. Bellman and L. A. Zadeh: *Decision-making in a fuzzy environment.* In: Management Science, 1970.
6. B. D. Anderson: *Failures of adaptive control theory and their resolution.* In: Communications in Information and Systems, Vol. 5, No. 1, pp. 1-20, 2005.
7. Niko Sünderhauf and O. Brock and W. Scheirer and Raia Hadsell and D. Fox and J. Leitner and B. Upcroft and P. Abbeel and W. Burgard and Michael Milford and P. Corke: *The limits and potentials of deep learning for robotics.* In: The International Journal of Robotics Research, 2018.
8. Gabriel Dulac-Arnold and N. Levine and Daniel J. Mankowitz and J. Li and Cosmin Paduraru and Sven Gowal and T. Hester: *An empirical investigation of the challenges of real-world reinforcement learning*, 2020.
9. J. Garcia and F. Fernández: *A comprehensive survey on safe reinforcement learning.* In: Journal of Machine Learning Research, 2015.
10. Mouhacine Benosman: *Learning-based adaptive control for nonlinear systems,* 2016.
11. T. Haarnoja, Aurick Zhou, P. Abbeel, S. Levine: *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.* In: International Conference on Machine Learning (ICML), 2018.
12. T. Chaffre and J. Moras and A. Chan-Hon-Tong and J. Marzat: *Sim-to-Real Transfer with Incremental Environment Complexity for Reinforcement Learning of Depth-Based Robot Navigation.* In: Proceedings of the 17th International Conference on Informatics in Control, Automation and Robotics, 2020.
13. Karl J. Astrom and Bjorn Wittenmark: *Adaptive Control, second edition.* In: Library of Congress Cataloging-in-Publication Data, published by DOVER PUBLICATIONS, INC, 2008.

14. Nikolai M. Filatov and H. Unbehauen: *Adaptive Dual Control Theory and Applications.* In: Lecture Notes in Control and Information Sciences, Vol. 302, published by Springer, 2004.
15. G. Stein: *Adaptive flight control: a pragmatic view.* In: Applications of Adaptive Control, eds. K. S. Narendra and R. V. Monopoli. New York: Academic Press, 1980.
16. Osburn, P. V. and H. P. Whitaker and A. Kezer: *New developements in the design of adaptive control systems.* In: Institute of Aeronautical Sciences, 1961.
17. U. Borisson: *Self-tuning regulators for a class of multivariable systems.* In: Automatica, 1979.
18. Sternby, J.: *A simple dual control problem with an analytical solution.* In: IEEE Transactions on Automatic Control, 1976.
19. A. Pyrkin and A. Bobtsov and S. Kolyubin and O. Borisov and Vladislav Gromov and S. Aranovskiy: *Output controller for quadcopters with wind disturbance cancellation.* In: IEEE Conference on Control Applications (CCA), 2014.
20. A. Razinkova and I. Gaponov and H.-C. Cho: *Adaptive control over quadcopter UAV under disturbances.* In : International Conference on Control, Automation and Systems (ICCAS), 2014.
21. R. S. Fernández and Sergio Domínguez and P. Campoy: *L1 adaptive control for Wind gust rejection in quad-rotor UAV wind turbine inspection.* In: International Conference on Unmanned Aircraft Systems (ICUAS), 2017.
22. C. Cao and N. Hovakimyan: *L1 Adaptive Output Feedback Controller to Systems of Unknown Dimension.* In: American Control Conference, 2007.
23. C. Cao and N. Hovakimyan: *Adaptive controller for systems with unknown time-varying parameters and disturbances in the presence of non-zero trajectory initialization error.* In: International Journal of Control, 2008.
24. S. Kim and Vedang M. Deshpande and R. Bhattacharya: $\mathscr{H}_2$ *Optimized PID Control of Quad-Copter Platform with Wind Disturbance.* In: International Conference on Unmanned Aircraft Systems (ICUAS), 2020.
25. Lucas M. Argentim and Willian C. Rezende and Paulo E. Santos and Renato A. Aguiar: *PID, LQR and LQR-PID on a quadcopter platform.* In: International Conference on Informatics, Electronics and Vision (ICIEV), 2013.
26. K. Ariyur and M. Krsti: *Real time optimization by Extremum Seeking control*, 2003.
27. Chunlei Zhang and R. Ord: *Extremum-Seeking Control and Applications: A Numerical Optimization-Based Approach*, 2011.
28. Andrea Tagliabue and Xiangyu Wu and M. Mueller: *Model-free Online Motion Adaptation for Optimal Range and Endurance of Multicopters.* In: International Conference on Robotics and Automation (ICRA), 2019.
29. R. Sutton and A. Barto: *Reinforcement Learning: An Introduction.* In: IEEE Transactions on Neural Networks, 2005.
30. Howard A. Ronald : *Dynamic Programming and Markov Processes.* In: The M.I.T. Press., 1960.
31. J. Schulman and P. Moritz and S. Levine and Michael I. Jordan and P. Abbeel: *High-Dimensional Continuous Control Using Generalized Advantage Estimation.* In: Computing Research Repository, 2016.
32. Antonio Loquercio and Ana I. Maqueda and Carlos R. del-Blanco and D. Scaramuzza: *DroNet: Learning to Fly by Driving.* In: IEEE Robotics and Automation Letters, 2018.
33. G. Kahn and A. Villaflor and B. Ding and P. Abbeel and S. Levine: *Self-Supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation.* In: IEEE International Conference on Robotics and Automation, 2018.

Direct Adaptive Pole-Placement Controller using Deep Reinforcement Learning: Application to AUV Control ⋆

Thomas Chaffre *,**,**** **Gilles Le Chenadec** *
Karl Sammut **,**** **Estelle Chauveau** ***
Benoit Clement *,**,****

* *Lab-STICC UMR CNRS 6285, ENSTA Bretagne, Brest, France (e-mail: name.surname@ensta-bretagne.fr).*
** *Centre for Maritime Engineering, Flinders University, Australia (e-mail: name.surname@flinders.edu.au).*
*** *Naval Group Research, Ollioules, France (e-mail: name.surname@naval-group.com).*
**** *CROSSING IRL CNRS 2010, Adelaide, Australia.*

Abstract: In this paper we investigate a direct adaptive learning-based tuning strategy for the control of an underwater vehicle under unknown disturbances. This process can be seen as a double integrator without delay and is usually regulated using a PD/PID type controller. A trade-off between performance and robustness may be found when tuning their parameters because a single optimal controller for multiple operating condition does not exist. Therefore, we use a re-parametrization of the PID controller gains in a space of poles where controller stability is guaranteed. We propose to use the maximum entropy deep reinforcement learning algorithm called SAC to explore this space. The adaptation procedure is able to capture a great variety of desired pole locations in order to adapt to process variations without measuring them. Simulation outcomes show the advantages of this approach.

Keywords: Adaptive control, Pole-placement, Deep reinforcement learning, Underwater vehicle.

1. INTRODUCTION

Operating in a constantly perturbed marine environment, autonomous underwater vehicles (AUVs) must compensate for wave and current induced forces acting on their body. In this context, a common practice is to employ a PID type control law with fixed parameters that are typically obtained using model-based optimization theory. Although PID controllers can be made to work reasonably well under static known environment conditions, the performance of the fixed parameter controller may diminish under adverse conditions when exposed to large dynamic variations in the process. Performance of PID regulators can however, be improved considerably by accommodating such changes using online PID regulator tuning techniques, such as adaptive control theory [O'Dwyer (2006)].

Adaptive control methods are widely used in the context of dynamic processes and provide, what seems to be, an ideal framework for automatic tuning of regulators. New tuning techniques are emerging from the development of data-driven theory [Brunton and Kutz (2019)]. The well-known model-free adaptive method, *Extremum Seeking* [Ariyur and Krstić (2003)], has been proven to compensate for uncharacterised and unmodelled process variations. Machine

⋆ This work was supported by ISblue project, Interdisciplinary graduate school for the blue planet (ANR-17-EURE-0015) and co-funded by a grant from the French government under the program "Investissements d'Avenir".

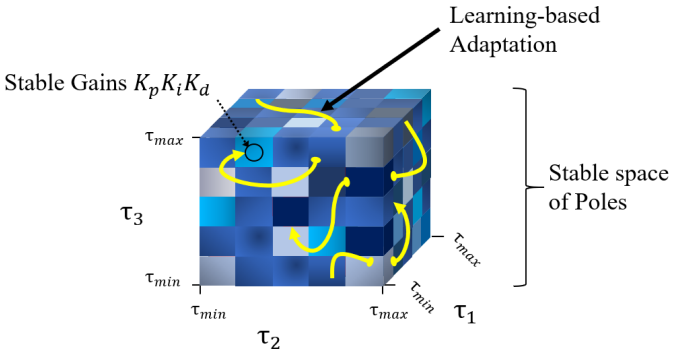


Fig. 1. Parameter cube representation for adaptive Pole-Placement control. The continuous space of possibles poles is restricted to stable locations. It is explored by a deep reinforcement learning algorithm (yellow lines) that feed these parameters to a PID controller to derive the gains and thus the control law. This guarantees adaptability while converging to the steady-state.

Learning based model-free adaptive methods have been suggested to tune PID regulators [Jayachitra and Vinodha (2014)]. Deep Reinforcement Learning techniques (DRL) in particular, have shown great performance as optimization methods for model-free adaptive scheme. They exploit the strong abilities of artificial neural networks (ANNs) to perform nonlinear mapping between sensor feedback and control inputs directly in closed-loop systems.

Nevertheless, the stability analysis remain difficult to derive in the ANN framework. There is no theoretical guarantee of performance and ANNs tends to be over parameterized which results in poor performance under real conditions. The contribution of this work can be summarized in two parts: **1.)** we propose a learning-based tuning procedure for the adaptive Pole-Placement of a PID controller which ensures stable poles selection despite using ANNs, and **2.)** we modified and extended the method that we originally designed for an aerial vehicle [Chaffre et al. (2021)] to the underwater environment and show the ease of applying it to a new class of process.

This paper is organized as follows: Section 2 reviews related work in DRL-based adaptive control for AUVs, Section 3 presents the simulated robotic platform and environment used, and in Section 4 a complete description of our approach is provided. The DRL-based tuning strategy is detailed in Section 5. Results obtained from testing the controller within a realistic simulation environment are provided in Section 6 to demonstrate the effectiveness of our proposed learning-based adaptive controller. In Section 7, the paper concludes with a discussion of the performance of the proposed approach.

2. RELATED WORK

In the underwater domain, process variation can occur in many forms and qualities [Fossen (2011)]. Among others, currents are often the dominants forces acting on an AUV. They are difficult to counter because they are caused by multiple phenomena mixing together such as: tidal movement, surface wind, heat exchange at the sea surface, salinity changes or density variation in higher depths. It is commonly assumed that sea currents vary slowly with time and therefore they are usually represented as a static and uniform force. This way, they can theoretically be compensated for by including an integral action.

Nevertheless, in practice the frequency of the current induced forces vary much more than assumed because they are also influenced by the velocity and heading of the vehicle. In addition, the motors and thrusters efficiency can also vary significantly during operation. It is therefore almost impossible to re-tune the control parameters using linear feedback control methods. The field of adaptive control theory has emerged as a solution to tackle this type of process. This broad family of techniques aims at allowing the control law to hold some flexibility by means of online parameters refinement. It has been used widely in the underwater context where the principal sources of disturbances (i.e. wind, waves and current) are mostly not observable. Recently, more concerted effort has been put in the design of adaptive controllers using DRL in order to adapt to a wide range of operating conditions.

In [Carlucho et al. (2018)], the Deep Deterministic Policy Gradient (DDPG) algorithm developed by [Lillicrap et al. (2016)] was used to teach an AUV how to perform a waypoint rallying mission. They took as inputs the position of the vehicle, its positioning error, its velocity and the last actions performed. The DDPG algorithm was then trained to directly map the thruster commands to these input variables in order to reach a waypoint. They designed a reward signal that is a function of the error of the distance to the target point to train the agent. Simulation results proved that the controller is able to complete the task even when facing unexpected fault scenarios (i.e. not seen during training). Indeed, the agent was able to complete the task even when facing a thrust reduction of 90% (despite undershooting the goal before reaching it in this case). These results showed the adaptive abilities of ANN-based adaptive control strategies.

The DDPG algorithm was later used in [Wang et al. (2018)] to perform adaptive trajectory planning for multiple AUVs with the added objective of satisfying constraints related to kinematics, communication range and sensing area. To do so, they designed a specific cost function and modified their *Experience Replay* approach. The first included terms that transcribe the field uncertainty, the cost of the trajectory and the constraints induced by the trajectory. In addition, they decided to store past experience of the agent (using the *Experience Replay* technique) that specifically satisfy the communication range and sensing area constraints in distinct replay buffers. The gradient updates are then applied to those transition that satisfy both requirements resulting in a policy that chooses actions that comply with the constraints. Simulation results showed that their approach is able to achieve a performance matching that of a benchmark method that assumes perfect knowledge of the field hyper-parameters.

From another design point of view, the DDPG algorithm was more recently combined with a Proportional-Derivative (PD) controller in [Knudsen et al. (2019)] for the station keeping of a Remotely Operated underwater Vehicle (ROV). Such controllers are denoted as learning-based adaptive controllers and can be seen as a mix of model-based and model-free control schemes. The hybrid controller was evaluated through simulations and real life experiments. The results proved that the hybrid controller is able to compensate for unknown external disturbances thanks to the DRL tuning while ensuring cautious control of the known part of the process with the PD controller.

As mentioned above, DRL techniques are increasingly being used in control and efforts are being made to combine them with classical control scheme, taking advantage of both model-based and model-free theory. Following this idea, we present a hybrid adaptive control strategy for the control of an AUV in the presence of unknown sea currents. In contrast to [Knudsen et al. (2019)], it is based on a PID controller. The gains of the PID controller can take a wide variety of values and their bounds are not trivial to choose for tuning. One could need many disparate values of gains in order to be overly conservative to satisfy many operating points. Some terms might also need to be removed at some point for safety or power conservation needs. Instead of using a DRL algorithm to directly explore the space of gains, our objective is to express them in a different space, as large as possible, where bounds and control performance criterion are easy to derive. For this purpose, the PID controller has been re-parametrized using a Pole-Placement strategy. It allows us to convert the gains in the poles domain. The poles are then tuned in this space by using the maximum entropy DRL algorithm called *Soft Actor-Critic* [Haarnoja et al. (2018)]. The resulting poles are then transformed back into the space of gains and the PID control law is finally applied as illustrated in Figure 1.

3. AUV SIMULATION AND MODELING

3.1 ROS packages

Because DRL algorithms are sample inefficient and their initial behavior can be unpredictable and possibly risky for robotic platforms, it is mandatory to perform their training in realistic simulators. If realistic enough, the simulator can theoretically provide an infinite amount of training data and the reality gap can be reduced with *Domain Randomization* [Tobin et al. (2017), Sadeghi and Levine (2017)]. For this purpose, we used the ROS [Quigley (2009)] package called *UUV Simulator* [Manhães et al. (2016)], to train our controller. This package provides a Gazebo-based simulation of underwater environments and the possibility to use the existing RexROV2 platform [Berg (2012)] as a test vehicle. In addition, it is also possible to simulate several disturbances including sea currents. They are simulated as an uniform force in the Gazebo world, represented by a linear velocity v_c (in $m.s^{-1}$), an horizontal h_c and vertical angle j_c (in radians).

3.2 AUV model

A ROV platform can be modelled using the general equations of motion for a marine craft, which can be written in the vectorial form according to [Fossen (1994)] as:

$$\begin{aligned} &\dot{\eta} = J_\Theta(\eta)\nu \\ &M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) = \delta + \delta_{cable} \end{aligned} \tag{1}$$

where η and ν are the position and velocity vectors respectively, δ is the control force vector and δ_{cable} is the vector describing the umbilical forces from the cable attached to the ROV. The RexROV2 is a ROV-type platform provided in UUV Simulator. It is propelled by 6 thrusters (complete details on its equation of motions are provided in [Berg (2012), McCue (2016), Yang et al. (2015)]). The control vector u is obtained by transforming the actuator force vector:

$$\delta = \mathbf{T}(\alpha)\mathbf{K}\mathbf{u} \tag{2}$$

where $\mathbf{T}(\alpha) \in \mathbb{R}^{n\times r}$ is the thrust allocation matrix ; $\mathbf{K}$ is the thrust coefficient matrix ; δ is the control force vector in nDOF and $\mathbf{u} \in \mathbb{R}^r$ is the actuator input vector. The package [Manhães et al. (2016)] allows us to setup a vector of thruster contribution for every DOFs (for clarity, $\sin(\cdot)$ and $\cos(\cdot)$ are denoted as $s\cdot$ and $c\cdot$ below):

$$\mathbf{T}_i = \begin{bmatrix} Surge \\ Sway \\ Heave \\ Roll \\ Pitch \\ Yaw \end{bmatrix} = \begin{bmatrix} c\theta c\phi \\ s\theta c\phi \\ s\phi \\ -Zs\theta + Ys\phi \\ -Zc\theta + Xs\phi \\ -Yc\theta + Xs\theta \end{bmatrix} \tag{3}$$

These vectors are then assembled into a thrust allocation matrix $\mathbf{T} = [T_1, \ldots, T_6]$ which decribes the relationship between propeller thrust and the vehicle speed (its calculation is available in [Carlton (2018)]). Using this allocation matrix, the control inputs u are transformed as $u = [v_x; v_y; v_z; \omega_\psi; \omega_\theta; \omega_\phi]^T$ where $[v_x; v_y; v_z]^T$ are linear velocity inputs (in $m.s^{-1}$) and $[\omega_\psi; \omega_\theta; \omega_\phi]^T$ are torque inputs (in radians) expressed in the reference frame attached to the center of mass of the simulated RexROV2 platform.

4. A PID CONTROLLER WITH DIRECT POLE-PLACEMENT

The control objective treated here is to control the AUV at a fixed velocity and orientation Λ_{ref} with $\dot{\Lambda}_{ref} = 0$. We define the error as $e = \Lambda_{ref} - \Lambda$. We take into account the wave-generated disturbance by considering the steady-state error variable $\sigma = \int_0^t e(\tau)d\tau$. The proposed control strategy is based on a classical PID applied on each axis. Unfortunately, the gain tuning is not easy and a reinforcement learning method is used to do it. In order to accelerate and make it easy to interpret, the PID controller is seen as a double integrator with a state-feedback. This gain mapping is then easier to tune as shown in Section 5.

4.1 State space system

The augmented model with the state vector $X = [\sigma, e, \Lambda]$ becomes

$$\frac{d}{dt}\begin{bmatrix}\sigma \\ e \\ \Lambda\end{bmatrix} = \underbrace{\begin{bmatrix}0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0\end{bmatrix}}_{\mathrm{A}}\begin{bmatrix}\sigma \\ e \\ \Lambda\end{bmatrix} + \underbrace{\begin{bmatrix}0 \\ 0 \\ 1\end{bmatrix}}_{\mathrm{B}} u \tag{4}$$

The PID control law is defined as

$$u = k_p e + k_i \sigma + k_d \Lambda \tag{5}$$

where $\max(k_i\sigma) = 2000$ for anti-windup compensation and $u(t) \in [-2000; +2000]$. We consider the control objective to be achieved if the value of each errors e_i on the setpoint Λ_{ref_i} stays around a predefined percentage χ of the desired value over a predefined time period of ς steps.

4.2 Pole-Placement strategy

The PID state-space representation is given by

$$\dot{X} = (A - BK)X \tag{6}$$

Among the various possible procedures that can be used to determine the gain K, a fundamental technique consists of assigning a set of specific values, $P = \{\lambda_1\ \lambda_2\ \ldots\ \lambda_n\}$, to the eigenvalues of the feedback loop $A - BK$. Given that these eigenvalues determine the poles of all the transmittance where the associated state matrices are involved, this procedure is denoted as Pole-Placement. We can define a (normalized) Control Polynomial as

$$C(s) = s^n + c_1 s^{n-1} + \cdots + c_{n-1}s + c_n \tag{7}$$

whose roots are the λ_i and assign it as the characteristic polynomial of $A - BK$ with

$$C(s) = det(s\mathbf{I} - (A - BK)) \tag{8}$$

Equations (6) and (8) yield

$$\begin{aligned} |A - BK - \lambda\mathbf{I}| &= \begin{vmatrix} -\lambda & 1 & 0 \\ 0 & -\lambda & 0 \\ -k_i & -k_p & -kd\lambda \end{vmatrix} \\ &= -\lambda\begin{vmatrix} -\lambda & 1 \\ -k_p & -kd - \lambda \end{vmatrix} - \begin{vmatrix} 0 & 1 \\ -k_i & -kd - \lambda \end{vmatrix} \end{aligned} \tag{9}$$

This can be rewritten as

$$\begin{aligned} |A - BK - \lambda \mathbf{I}| &= -\lambda(\lambda(k_d + \lambda) + k_p) - k_i \\ &= -\lambda^3 - \lambda^2 k_d - \lambda k_p - k_i \\ &= 0 \end{aligned} \tag{10}$$

This is true if and only if

$$\lambda^3 + \lambda^2 k_d + \lambda k_p + k_i = 0 \tag{11}$$

In order to maintain controller stability, the poles of (11) must be placed in the left half-plane (i.e. $A - BK$ is Hurwitz) which ensure convergence of the controller to steady-state without oscillations. For this purpose, the poles of the controller (5) must be solutions of the equation (11). The desired poles $\tau_1 > 0, \tau_2 > 0, \tau_3 > 0$ are then defined as

$$\lambda_1 = \frac{-1}{\tau_1}\,;\, \lambda_2 = \frac{-1}{\tau_2}\,;\, \lambda_3 = \frac{-1}{\tau_3} \tag{12}$$

The Pole-Placement design can be written as

$$\begin{cases} \frac{-1}{\tau_1^3} + \frac{k_d}{\tau_1^2} - \frac{k_p}{\tau_1} + k_i = 0 \\ \frac{-1}{\tau_2^3} + \frac{k_d}{\tau_2^2} - \frac{k_p}{\tau_2} + k_i = 0 \\ \frac{-1}{\tau_3^3} + \frac{k_d}{\tau_3^2} - \frac{k_p}{\tau_3} + k_i = 0 \end{cases} \tag{13}$$

Since τ_1, τ_2 and τ_3 are solutions to (11), it follows that

$$\begin{bmatrix} 1 & \frac{-1}{\tau_1} & \frac{1}{\tau_1^2} \\ 1 & \frac{-1}{\tau_2} & \frac{1}{\tau_2^2} \\ 1 & \frac{-1}{\tau_3} & \frac{1}{\tau_3^2} \end{bmatrix} \begin{bmatrix} k_i \\ k_p \\ k_d \end{bmatrix} = \begin{bmatrix} \frac{1}{\tau_1^3} \\ \frac{1}{\tau_2^3} \\ \frac{1}{\tau_3^3} \end{bmatrix} \Leftrightarrow MK^T = N \tag{14}$$

Finally, the gains of the control law (5) are obtained with $K^T = M^{-1}N$ as

$$\begin{aligned} k_i &= \frac{1}{\tau_1 \tau_2 \tau_3} \\ k_p &= \frac{\tau_1 + \tau_2 + \tau_3}{\tau_1 \tau_2 \tau_3} \\ k_d &= \frac{\tau_1 \tau_2 + \tau_1 \tau_3 + \tau_2 \tau_3}{\tau_1 \tau_2 \tau_3} \end{aligned} \tag{15}$$

This mapping allows optimisation of the exploration space.

4.3 Choice of closed-loop poles

The performance of a Pole-Placement controller is directly related to its poles location which make their tuning a critical task. There exist a set of fundamental design rules to guide this choice [Chilali and Gahinet (1996)] but in general one should not choose closed loop poles that are highly negative, making the system fast responding (in the frequency domain) which leads to large bandwidth and thus amplification of noise.

In our case, with the design (12), for any value of $\tau_i > 0$, the poles of the controller are placed in the left half-plane which ensure its stability. This leaves us with the settling time requirement to define in order to bound the value of the poles. In accordance with the control objective (4.1), we define the desired settling time of the close-loop control $\varsigma = 10$ seconds as the time after which we want the system outputs to stay within $\chi = 2.5\%$ around its desired values. The upper bound of the poles is derived as

$$\lambda_{max} = \frac{\ln(\chi)}{\varsigma} = \frac{\ln(0.025)}{10} = -0.368 \tag{16}$$

Therefore $\lambda_i \leqslant \lambda_{max}$ (12), from which we can derive the upper bound of the poles

$$\tau_{min} < \tau_i \leqslant \frac{1}{-\lambda_{max}} = 2.710 = \tau_{max} \tag{17}$$

We choose $\tau_{min} = 0.1$ because for lower values the control inputs are too expensive in terms of control efforts and too aggressive for our control objective. Thus, the bounds of the poles are defined as

$$0.1 \leq \tau_i \leq 2.710 \tag{18}$$

There exist a solution for all $C(s)$ (7) if and only if the pair (A,B) is controllable, which is assumed here. In the case of a Single-Input system, the solution is unique. In the case of Multi-Input systems as studied here, the number of free component of the matrix K is greater than the n eigenvalue constraints. Accordingly, there exist an infinite number of solutions, among which it is not trivial to define an *Optimal* solution. In fact, depending on the process dynamics, we might favor one particular configuration of pole locations in the space (18) over another. The Linear Quadratic (LQ) optimization scheme then composes an alternative framework to define optimal values.

In contrast, we propose to use DRL to adapt these parameters for the control of an AUV, which consist in searching for the best values possible within (18) based on the process measurements. More precisely, our approach consist in using a Deep Policy Gradient method [Sutton and Barto (2005)] whose objective is to explore the space of poles (18) in order to find at each time step the best values possible (in this space) for each control input. The policy objective is to adjust them based on the process variation and with respect to a reward function which emphasizes the control objective. This is denoted as a *Direct* method because the controller parameters are adjusted directly without the need for an estimation of any process parameters. With this mapping (15), the learning action space is limited to desired solutions in the poles space. This ensure that for any pole values estimated by the ANN, the resulting control law will maintain the poles of the controller in the left half-plane.

This adaption is performed in closed loop and therefore exploits an extensive state representation of the process. We believe that this is highly beneficial to the parameters adaptation which is hence able to capture a large variety of appropriate poles. In the well known original Direct Adaptive Pole-Placement (DAPP) suggested by H. Elliott [Elliott (1981), Elliott et al. (1982)], there is neither an explicit reference model nor an error beteen the reference model output and system. The desired dynamical behavior is used implicitly in the process of indirect identification. In our version of DAPP, there is no parameters identification and the desired behavior is solely characterize by a *reward signal*, making its design a critical task.

5. REINFORCEMENT LEARNING FRAMEWORK

Reinforcement Learning (RL) is a subclass of Machine Learning methods where a learning agent that is evolving in an environment has to learn how to take optimal actions. The agent interacts with its environment by executing at each time step action $a_t \in A$ from state $s_t \in S$ which makes it transit to a new state $s_{t+1} \in S$. This transition produces a scalar value $r_t \in \mathbb{R}$ known as the reward. The sequence of T actions and $T+1$ states in the environment may be framed as a Markov Decision Process [Howard (1960)] along a trajectory $\zeta = (s_0, a_0, s_1, a_1, \ldots, a_{T-1}, s_T)$. This stochastic process is completely defined by $p(\zeta)$ involving the initial probability $\boldsymbol{s}_0 \sim p(\boldsymbol{s}_0)$, the policy $\pi_\mu : S \rightarrow A$ parameterized by μ and the state transition probability $p(s_{t+1}|s_t, a_t)$. The optimization problem historically considered in RL is to estimate the policy $\pi^\star$ which maximizes the expected return while following this policy:

$$\pi^\star = \arg\max J_{RL}(\pi_\mu) \tag{19}$$

where $J_{RL}(\pi_\mu)$ denotes the expected return along the trajectory ζ as:

$$J_{RL}(\pi_\mu) = \mathop{\mathbb{E}}_{\zeta\sim p(\zeta)}\left[\sum_{t=0}^{T-1} r_t\right] = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\mu}} r(s_t, a_t) \tag{20}$$

where ρ_{π_μ} is the state-action joint probability. In this article, we propose to use the *Policy Gradient* algorithm [Sutton and Barto (2005)] called *Soft Actor-Critic* which aims at modeling and optimizing the policy directly.

It is composed of three key components:

(1) An Actor-Critic architecture [Konda and Tsitsiklis (1999)] with separate values and policy networks.
(2) An off-policy formulation that enable the use of past collected data with Experience Replay [Lin (1992)].
(3) Entropy maximization for improved stability and exploration [Haarnoja et al. (2017)].

5.1 Soft Actor-Critic

The learning objective is here to estimate a policy π_μ that directly maps the pole locations from the AUV state in order to reach the control objective:

$$\pi_\mu : \; s_t \rightarrow [\; \tau_{vx} \; ; \; \tau_{vy} \; ; \; \tau_{vz} \; ; \; \tau_{roll} \; ; \; \tau_{pitch} \; ; \; \tau_{yaw} \;] \tag{21}$$

where $dim(\tau_i) = 3$, thus the dimension of the action space is 18. These poles candidates are then applied to compute the gains (15) that are used to derive the control law (5) for each control inputs. For this purpose, we used the algorithm named Soft Actor-Critic. It is denoted as a DRL method because the parameter μ is estimated by an ANN. Contrary to the classic DRL optimization objective (19), the Soft Actor-Critic aims at maximizing the expected return as well as the entropy of the policy:

$$J_{SAC}(\pi_\mu) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\mu}}[r(s_t, a_t) + \alpha H(\pi_\mu(.|s_t))] \tag{22}$$

where

$$H(\pi_\mu(.|s)) = -\sum_{a\in A} \pi_\mu(a) \log \pi_\mu(a|s) \tag{23}$$

The term $H(\pi_\mu)$ is the entropy measure of policy π_μ and α is a fixed temperature parameter that determines the relative importance of the entropy term (i.e. the trade-off between reward maximization and entropy maximization). By doing this, the policy is forced to explore sub-optimal solutions (i.e. actions associated to similar high Q-Values) until it identifies which ones are really better for the long term maximization objective. This has been proven to greatly improve the sampling efficiency of off-policy DRL techniques [Haarnoja et al. (2017)]. The SAC method includes estimation of the Q-Value and State-Value functions by ANNs whose parameters w and Ψ are respectively updated to minimize the errors [Haarnoja et al. (2018)]:

$$\begin{aligned} J_Q(w) =& \mathbb{E}_{(s_t,a_t)\sim D}\Bigg[\frac{1}{2}\Bigg(Q^{\pi_\mu}_w(s_t, a_t) - \Big(r(s_t, a_t) \\ &+ \gamma \mathbb{E}_{s_{t+1}\sim\rho_\pi(s)}[V^{\pi_\mu}_{\bar{\Psi}}(s_{t+1})]\Big)\Bigg)^2\Bigg] \end{aligned} \tag{24}$$

$$\begin{aligned} J_V(\Psi) = \mathbb{E}_{s_t\sim D}\Big[\frac{1}{2}(V_\Psi(s_t) - \mathbb{E}[Q^{\pi_\mu}_w(s_t, a_t) \\ - \log \pi_\mu(a_t, s_t)])^2\Big] \end{aligned} \tag{25}$$

where w and $\bar{\Psi}$ respectively, denote the parameters of the Q-Value and State-Value networks. The ANN estimated parameters μ of the policy function are updated in order to minimize the expected Kullback-Leibler divergence:

$$J_\pi(\mu) = \mathbb{E}_{s_t\sim D}\left[D_{KL}\left(\pi_\mu(\cdot|s_t)\middle\|\frac{\exp(Q^{\pi_\mu}_w(s_t, \cdot))}{Z^{\pi_\mu}_w(s_t)}\right)\right] \tag{26}$$

5.2 State vector

At each time step, the agent captures an observation vector of the process o_t as described below:

$$o_t = [\; a_{t-1} \; ; \; O \; ; \; V \; ; \; \Omega \; ; \; u_i \; ; \; e_i \; ; \; e_{L_2} \;] \tag{27}$$

where $a_{t-1}\in\mathbb{R}^{18}$ are the estimated pole values (21); the Euler orientation of the vehicle are $O = [\phi, \theta, \psi]$; its linear and angular velocities are respectively $V = [v_x, v_y, v_z]$ and $\Omega = [\omega_\phi, \omega_\theta, \omega_\psi]$; the vector $u_t \in \mathbb{R}^6$ is composed of the current PID controller outputs, $e_i \in \mathbb{R}^6$ are the errors on each setpoint and e_{L_2} is the Euclidean distance to the steady-state as in (30). Due to latencies and uncertainties in the process, the dynamics can become non-Makorvian which significantly degrade the learning performance. For this reason, we construct the state vector out of the current and past 4 observation vectors, thus $dim(s_t) = 200$.

5.3 Artificial neural network architecture

Five multilayer perceptron networks are trained in order to minimize the above loss functions (24)-(26). Four of these include a Policy network, two Q-value networks and a State-Value network. They respectively take as input, the state (s_t), the pair of state and actions (s_t, a_t), and the state (s_t) vectors. The two Q-Value networks are used to reduce the overestimation bias of Actor-Critic methods [Hasselt et al. (2016), Fujimoto et al. (2018)]. The minimum between the two Q-Value estimates is used to compute the losses of the State-Value (25) and Policy

(26) networks. A fifth network, a separate target State-Value network that slowly tracks the actual State-Value network is employed and updated using an exponentially moving average with a smoothing constant $\Upsilon = 5 \times 10^{-3}$ as in [Lillicrap et al. (2016), Mnih et al. (2015)].

The networks are trained simultaneously using Gradient descent and Adam [Kingma and Ba (2015)] as optimizer. Each network is composed of two hidden layers of 256 hidden units. The ReLU6 [Krizhevsky (2010)] activation function is applied to all the hidden layers, Tanh is applied to the output layer of the Policy network while no activation functions are applied to the final layers of the critics. The learning rate is fixed and equal to 3×10^{-4} for all networks [Haarnoja et al. (2018)] and with $\gamma = 0.99$ for (24). The following addition to the first version of the SAC algorithm has been made:

(1) Because value estimates tend to overestimate when the policy is poor, and the policy tends to be poor if the value estimate itself is inexact, we delay their updates by updating the target State-Value and Policy networks after a fixed number of updates $d = 2$ to the critics as proposed in [Fujimoto et al. (2018)].
(2) Batch normalization [Ioffe and Szegedy (2015)] is used before the ReLU6 layers of each networks as suggested in [Lillicrap et al. (2016)].
(3) To remove dependency on the initial weights, we use a random policy for the first $50,000$ time steps. Then, we add to each action a Gaussian noise $\mathcal{N}(0, 01)$.

Mini-batch gradient descent is applied at each time step with a mini-batch size of 256 past transitions, randomly sampled from a replay buffer of maximum size 10^6 and with the CER technique [Zhang and Sutton (2017)].

5.4 Reward function

In accordance with the control objective, the following reward function is introduced:

$$r(s_t) = \begin{cases} r_{succeed} \textbf{ if } \forall t \in [t-100; t],\ |e_i(t)| \leq \chi, \\ r_{regulation}\textbf{, otherwise.} \end{cases} \tag{28}$$

The reward signal $r_{regulation}$ is a binary reward signal defined as:

$$r_{regulation} = \begin{cases} 20 \times e^{-(e_{L_2})^2} \times (1 + dt_{e_i(t)}) \textbf{ if } dt_{e_i(t)} > 0, \\ -20\textbf{, otherwise.} \end{cases} \tag{29}$$

where e_{L_2} is the Euclidean distance to the steady-state:

$$e_{L_2} = \sqrt{\sum_{i=1}^{i=dim(u)} e_i^2(t)} \tag{30}$$

and $dt_{e_i(t)}$ is the derivative of the error e_i computed over two time steps such as:

$$dt_{e_i(t)} = e_i^2(t-1) - e_i^2(t) \tag{31}$$

This reward function highly recompenses error decrease and penalizes any deviation from the Euclidean path to the steady-state. If the control objective is met, the reward $r_{succeed} = 500$ is generated which ends the current episode.

5.5 Training scenario

The training consisted in performing a total of 1 million time step iterations. Each episode has a maximum length of 300 time steps (equivalent to 20 seconds). This parameter also required tuning because depending on the problem, an overflow of steps can highly degrade and slow the learning, while too few steps often cause the policy to over-fit. A training episode is defined as follow:

(1) At the beginning of the episode the AUV is initialized at the position $(x_0, y_0, z_0) = (0, 0, -40)$ with null velocity and a random orientation $(\psi_0, \theta_0, \phi_0) \in [\frac{-\pi}{4}; \frac{\pi}{4}]$.
(2) A random set of current variables is generated such as $v_c \in [0, 0.5]$ and $[h_c, j_c] \in [\frac{-\pi}{4}; \frac{\pi}{4}]$ which are then kept constant during the episode.
(3) A random vector of setpoints is generated such that $\Lambda_{ref} = [v_x, 0, 0, 0, 0, 0]^T$ with $v_x \in [0.5, 1.5]$.
(4) Then, the off-policy exploration strategy is used and the episode ends when the step number reaches 300 or $r_{succeed}$ is generated.

The PyTorch framework [Paszke et al. (2019)] was used along with the CUDA toolkit [Nickolls et al. (2008)] and an RTX 2070 GPU card, allowing us to perform the training in approximately 18 hours. Note that the Gazebo simulation is run here at a real time factor, therefore this training time could be further reduced.

5.6 Evaluation scenarios

We defined three evaluation scenarios based on the characteristics of the process:

- Scenario 1: the setpoints are fixed during the whole episode while the sea current variables vary.
- Scenario 2: the sea current variables are fixed during the whole episode while the setpoints vary.
- Scenario 3: both sea current variables and setpoints vary during the episode.

When varying, these variables have the form:

$$[v_x, h_c, j_c, v_c] = D_1 \times sin(D_2 \times t) + D_3 \tag{32}$$

with $D_1 \in [0.1; 0.25]$, $D_2 \in [0.5; 1]$ and $D_3 \in]1; 1.25]$ for $[v_x]$ and $D_1 \in [0.25; 1]$, $D_2 \in [0.5; 1]$ and $D_3 \in]0; 0.5]$ for $[h_c, j_c, v_c]$ randomly chosen. The evaluation consists in performing 500 episodes for each scenario with a maximum step size per episode of 500 and different from each other in terms the above mentioned variables.

6. SIMULATION RESULTS

6.1 Evaluation outcomes

As a benchmark, we used the optimal PID controller provided in the UUV Simulator to show the benefits of our approach. It is a 6-DoF PID controller whose parameters have been optimized using SMAC [Hutter et al. (2011)]. Both controllers are therefore based on the exact same structure (PID type control law) for a fair comparison. Evaluation outcomes are provided in the table 1. To compare the controllers performance, we use the following metrics: the success rate, the mean reward per step and

Controller performance				
Scenario number	Controller type	Success rate	Mean reward per step	Positive reward rate
1	Optimal PID	0.22	9.398	0.753
	DRL-DAPP	0.88	15.139	0.917
2	Optimal PID	0.07	-1.226	0.503
	DRL-DAPP	0.79	12.11	0.872
3	Optimal PID	0.05	0.878	0.540
	DRL-DAPP	0.56	6.2	0.810

Table 1. Simulation results.

the positive reward rate (i.e. the rate of actions that made the error decreased) computed over all episodes for each scenario. We can see that the optimal PID is not able to complete the task despite its ability at keeping the error very low. The numerous variation in the process makes the task of maintaining the error within a threshold extremely hard for a controller with fixed parameters. Our method, on the other hand demonstrates better results with a higher success rate and mean reward per step especially for scenario 1 and 2. However, the performance of our controller is limited on scenario 3 with a success rate almost equal to 50%. This could be explained by the fact that during training, the robot was facing constant currents and setpoints only and therefore failed to build a policy that can complete the task when they both vary.

7. CONCLUSIONS

In this paper we have presented a direct tuning scheme for the adaptive control of an AUV under unobservable disturbances. We proposed a new application of the SAC algorithm to perform adaptive Pole-Placement. Although the stability analysis is limited to the controller, our mapping provides some guarantees on the outputs of the neural network, which is often highlighted by the control community as a major concern toward the use of such methods in the real world. Finally, we demonstrated the benefits of our approach with simulated comparison to the model-based optimal counterpart of our control structure.

ACKNOWLEDGEMENTS

This study was supported by SENI, the joint research laboratory between NAVAL GROUP and ENSTA Bretagne.

REFERENCES

Ariyur, K.B. and Krstić, M. (2003). Real time optimization by extremum seeking control.

Berg, V. (2012). Development and Commissioning of a DP system for ROV SF 30k. Ph.D. thesis, NTNU.

Brunton, S.L. and Kutz, J.N. (2019). Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control. Cambridge University Press.

Carlton, J. (2018). Marine propellers and propulsion. Butterworth-Heinemann.

Carlucho, I., Paula, M.D., Wang, S., menna, B., Petillot, Y., and Acosta, G. (2018). Auv position tracking control using end-to-end deep reinforcement learning. OCEANS 2018 MTS/IEEE Charleston, 1–8.

Chaffre, T., Moras, J., Chan-Hon-Tong, A., Marzat, J., Sammut, K., Chenadec, G.L., and Clement, B. (2021). Learning-based vs model-free adaptive control of a MAV under wind gust. ArXiv, abs/2101.12501.

Chilali, M. and Gahinet, P. (1996). H_∞ design with pole placement constraints: an lmi approach. IEEE Transactions on Automatic Control, 41, 358–367.

Elliott, H. (1981). Direct adaptive pole placement with application to nonminimum phase systems. 1981 20th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes, 531–536.

Elliott, H., Wolovich, W., and Das, M. (1982). Arbitrary adaptive pole placement for linear multivariable systems. 1982 21st IEEE Conference on Decision and Control, 260–265.

Fossen, T. (1994). Nonlinear Modelling And Control Of Underwater Vehicles. Ph.D. thesis, NUST.

Fossen, T.I. (2011). Handbook of Marine Craft Hydrodynamics and Motion Control. Wiley.

Fujimoto, S., Hoof, H.V., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. ArXiv, abs/1802.09477.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. In ICML.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the 35th International Conference on Machine Learning, 1861–1870. PMLR.

Hasselt, H.V., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In AAAI.

Howard, R.A. (1960). Dynamic programming and markov processes. John Wiley.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In LION.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. ArXiv, abs/1502.03167.

Jayachitra, A. and Vinodha, R. (2014). Genetic algorithm based pid controller tuning approach for continuous stirred tank reactor. Adv. Artif. Intell., 2014.

Kingma, D.P. and Ba, J. (2015). Adam: A method for stochastic optimization.

Knudsen, K.B., Nielsen, M.C., and Schjølberg, I. (2019). Deep learning for station keeping of auvs. In MTS/IEEE OCEANS. Seattle.

Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. In NIPS.

Krizhevsky, A. (2010). Convolutional deep belief networks on cifar-10.

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. CoRR, abs/1509.02971.

Lin, L. (1992). Reinforcement learning for robots using neural networks.

Manhães, M., Scherer, S., Voss, M., Douat, L., and Rauschenbach, T. (2016). UUV simulator: A gazebo-based package for underwater intervention and multi-robot simulation. In MTS/IEEE OCEANS. Monterey.

McCue, L. (2016). Handbook of marine craft hydrodynamics and motion control [bookshelf]. IEEE Control Systems Magazine, 36(1), 78–79.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518, 529–533.

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with CUDA. 2008 IEEE Hot Chips 20 Symposium (HCS), 1–2.

O'Dwyer, A. (2006). Handbook of PI and PID controller tuning rules. Imperial College Press.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In NeurIPS.

Quigley, M. (2009). Ros: an open-source robot operating system. In ICRA 2009.

Sadeghi, F. and Levine, S. (2017). CAD^{2}RL: Real single-image flight without a single real image. ArXiv.

Sutton, R. and Barto, A. (2005). Reinforcement learning: An introduction. IEEE Transactions on Neural Networks, 16, 285–286.

Tobin, J., Fong, R.H., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 23–30.

Wang, C., Wei, L., Wang, Z., Song, M., and Mahmoudian, N. (2018). Reinforcement learning-based adaptive trajectory planning for auvs in under-ice environments. In MTS/IEEE OCEANS. Charleston.

Yang, R., Clement, B., Mansour, A., Li, M., and Wu, N. (2015). Modeling of a complex-shaped underwater vehicle for robust control scheme. Journal of Intelligent and Robotic Systems.

Zhang, S. and Sutton, R. (2017). A deeper look at experience replay. ArXiv, abs/1712.01275.

Preprints, IFAC CAMS 2022
Kgs. Lyngby, Denmark. September 14-16, 2022

PID Tuning using Cross-Entropy Deep Learning: a Lyapunov Stability Analysis

Hector Kohler [*] **Benoit Clement** [**,***] **Thomas Chaffre** [*,**] **Gilles Le Chenadec** [*]

[*] *Lab-STICC UMR CNRS 6285, ENSTA Bretagne, Brest, France (e-mail: name.surname@ensta-bretagne.fr).*
[**] *Centre for Maritime Engineering, Flinders University, Australia (e-mail: name.surname@flinders.edu.au).*
[***] *CROSSING IRL CNRS 2010, Adelaide, Australia.*

Abstract: Underwater Unmanned Vehicles (UUVs) have to constantly compensate for the external disturbing forces acting on their body. Adaptive Control theory is commonly used there to grant the control law some flexibility in its response to process variation. Today, learning-based (LB) adaptive methods are leading the field where model-based control structures are combined with deep model-free learning algorithms. This work proposes experiments and metrics to empirically study the stability of such a controller. We perform this stability analysis on a LB adaptive control system whose adaptive parameters are determined using a Cross-Entropy Deep Learning method.

Keywords: Underwater Vehicle, Adaptive Control, Deep Learning, Lyapunov Stability.

1. INTRODUCTION

Operating in a constantly disturbed environment, Unmanned Underwater Vehicles (UUVs) must compensate for wave and current-induced forces acting on their body. In this context, a common practice is to exploit a Proportional-Integral-Differential (PID) control law, whose parameters are obtained generally using optimal model-based control theory and are then kept constant during operation. However, their performances decay under intensive process variation and uncertainties such as those found in the underwater environment, where the vehicle state measurements are limited and the uncertainty on the external disturbance is high. The performance of PID regulators can be improved by accommodating such changes using online tuning techniques, such as adaptive control theory O'Dwyer (2006); Chaffre et al. (2021, 2022).

Adaptive control methods are widely used in the context of dynamical systems and provide what seems to be an ideal framework for automatic tuning of regulators Åström and Wittenmark (2013); Parks (1981). New tuning techniques are emerging from the development of data-driven theory Brunton and Kutz (2019), with Genetic algorithm also suggested to tune PID regulators Jayachitra and Vinodha (2014). Deep Reinforcement Learning techniques (DRL) in particular, have shown great performance as optimization methods. They exploit the strong representation abilities offered by artificial neural networks (ANNs) to build non-linear mapping functions between sensor feedback and control inputs/controller parameters, directly in closed-loop systems. Such methods are denoted as learning-based (LB). Despite their increased performance, their usage in UUV applications, where process observability is low, is still limited due to the stability analysis being hardly practicable to conduct when using ANNs.

In this work, we propose an adaptive LB controller where the PID gains are automatically adjusted by an ANN for the task of minimizing setpoint tracking error (i.e. position and orientation) of UUVs under Lyapunov stability constraints. The ANN is trained with a Cross-Entropy Method (CEM) Rubinstein (1997) which, unlike gradient-based optimization methods, is a direct policy search technique. Moreover, the considered application allows us to apply the classic Lyapunov stability analysis.

This paper is organized as follows: Section 2 presents some related usage of Deep Learning methods for the control of UUVs. In Section 3, we present the considered robotic platform with its modeling and the simulation framework we used to train our controller. Our strategy for stability analysis is presented in Section 4. In Section 5 we describe our learning-based (LB) PID tuning approach. A complete description of our usage of Deep Learning is provided in Section 6 with the complete hyperparameters choice. The experiments are described in Section 7 with an analysis of the results, leading to some open questions.

2. RELATED WORK

Numerous aspects of PI-PID tuning are discussed in O'Dwyer (2006). The most common method consists of auto-tuning where an experiment is performed in open or closed-loop to estimate the process model by using recursive least squares. The idea is that if a second-order model can be generated, it can then be used to make optimal pole-placement. However, the underwater context does not allow such procedures as open-loop experiments are too dangerous for the vehicles and the limited observability of the system/environment does not allow satisfying system identification. Recently, Deep Learning methods have been successively used in the field of PID tuning.

Among the various existing techniques, Deep Reinforcement Learning (DRL) Sutton and Barto (2018) has shown great performance in the adaptive control of UUVs Wang et al. (2018), Knudsen et al. (2019). These methods are used to iteratively build an estimate of an optimal policy function, mapping the process states to the control parameters. The DRL-based methods oblige to frame the control process as a Markov Decision Process (MDP). The associated Bellman's equations are then estimated using ANNs, usually involving Gradient-based learning. In contrast, we proposed in this paper to use CEM to optimize the ANN, removing the need for the MDP formulation.

3. AUV MODELING AND SIMULATION

3.1 AUV model

A mathematical model of a ROV platform can be derived using the general equations of motion for a marine craft, which can be written in the vectorial form according to Fossen (2011) as:

$$\begin{cases} \dot{\eta} = J_{\Theta}(\eta)\nu \\ M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) = \delta + \delta_{cable} \end{cases} \tag{1}$$

where η and ν are the position and velocity vectors respectively, δ is the control force vector and δ_{cable} is the vector describing the umbilical forces from the cable attached to the ROV. The RexROV2 is an ROV-type platform provided in UUV Simulator. It is propelled by 6 thrusters (complete details on its equation of motions are provided in Berg (2012), McCue (2016), Yang et al. (2015). The control vector u is obtained by transforming the actuator force vector: $\delta = T(\alpha)Ku$, where $T(\alpha) \in \mathbb{R}^{n\times r}$ is the thrust allocation matrix; K is the thrust coefficient matrix; δ is the control force vector in n degrees of freedom (DoF) and $u \in \mathbb{R}^r$ is the actuator input vector. The package Manhães et al. (2016) allows us to design a vector of thruster contribution for every DoFs.

3.2 ROS packages

We used the ROS-based package called *UUV Simulator* proposed in Manhães et al. (2016), to train our controller. This package provides a Gazebo-based simulation of underwater environments and the possibility to use the existing RexROV2 platform described in Berg (2012) as a test vehicle. In addition, it is also possible to simulate several disturbances including sea currents. They are simulated as a uniform force in the Gazebo world and represented by a linear velocity v_c (in $m.s^{-1}$), a horizontal h_c, and a vertical angle j_c (in radians).

4. OUR LEARNING-BASED ADAPTIVE CONTROL FRAMEWORK

In this paper, we address the problem of setpoint regulation where the UUV's objective is to minimize its tracking error in all 6 DoFs. The tracking error e is defined as the error between, the current UUV's position and orientation, and a fixed desired position and orientation as:

$$e = \eta_d - \eta. \tag{2}$$

The position and orientation of the UUV is:

$$\eta = (x, y, z, roll, pitch, yaw), \tag{3}$$

and the desired setpoint:

$$\eta_d = (0, 0, 0, 0, 0, 0). \tag{4}$$

In our LB adaptive control framework, a parametrization such as an artificial neural network, of the PID's gains is learned (in simulation or in the real world). In the LB control framework's testing phase, at every state measurement, a control step is performed. In our case, during a control step, the state of the control process is used as input of a trained neural network. The outputs of the network parametrize the PID gains (see section 5) used to compute the control input: the UUV's thrusters. The PID control law is defined as the following:

$$u = B^{-1}[\ J^T(\eta)\big(k_p e + k_i \int_0^t e(\tau)d\tau - k_d\dot{\eta}\big) + g(\eta)\ ], \tag{5}$$

where $u \in \mathbb{R}^6$ are the control inputs; the PID gain matrices $k_p, k_i, k_d \in \mathbb{R}^{6\times6}$ are the output of a neural network; $\eta \in \mathbb{R}^6$ is the current UUV's state, i.e. position and orientation (Eq. 3); $\dot{\eta} \in \mathbb{R}^6$ is the temporal derivative of the state vector; $g(\eta)$ is the sum of external forces acting on the UUV's body (in our case, gravity); $e \in \mathbb{R}^6$ is the tracking error (Eq. 2); B^{-1} is the fixed known thrusters allocation matrix mapping the control input u into a combination of thruster power inputs resulting to the desired movement and $J^T(\eta)$ is a transformation matrix.

The principal advantage of the considered PID regulator (Eq. 5) is that global stability and convergence analysis of the system has been well formalized by Fossen Fossen (1994). Lyapunov stability theory Liberzon (2005) is straightforward to apply with such a control law. Following Fossen (1994), there exists a Lyapunov function for the considered UUV such that:

$$V(x) = \frac{1}{2}x^T \begin{bmatrix} M_\eta^{-1} & \alpha I & 0 \\ \alpha I & k_p & k_i \\ 0 & K_i & \alpha k_i \end{bmatrix} x, \tag{6}$$

where $\alpha \in \mathbb{R}$ is a small positive constant and the PID gains are $k_p, k_i, k_d \in \mathbb{R}^{6\times6}$; M_η^{-1} is related to the UUV's mass and can be computed from the current η; x is the control loop's state (not to be mistaken with the UUV's state): $x = \left[p, \eta, \int_0^t e(\tau)d\tau\right]^T \in \mathbb{R}^{18}$ and $p = M_\eta\dot{\eta}^T \in \mathbb{R}^6$ is the generalized momentum depending on the UUV's mass and velocity. The control loop is stable at state x if:

$$V(x) > 0 \text{ and } \dot{V}(x) < 0. \tag{7}$$

Following Lyapunov stability theory Liberzon (2005), there exists theoretical constraints on the gain matrices k_p, k_i, k_d and the small constant α such that local stability is guaranteed when the initial conditions of the systems are closed to $\boldsymbol{x} = 0$. According to the proposed Lyapunov function (Eq. 6), the vehicle stability and convergence to steady-state are guaranteed (Eq. 7) if the following constraints are satisfied:

$$\begin{cases} k_d > M_\eta, \\ k_i > 0, \\ k_p > k_d + \frac{2}{\alpha}k_i, \\ \frac{1}{2}(1-\alpha)k_d - \alpha M_\eta + \frac{\alpha}{2}\sum_{i=1}^{6}(\eta_i - \eta_{id})\frac{\partial M_\eta}{\partial \eta_i} > 0, \\ \alpha > 0, \end{cases} \tag{8}$$

We found that, when limiting the parameter space to value satisfying (Eq. 8), the resulting space is so small that the benefits of adaptive control are merely preserved. Therefore, we propose not to take into account the stability constraints in the parameters optimization. Our objective is then to assess to what extent the resulting solution, obtained from an ANN, can still hold some stability components. In order to facilitate the stability analysis, we aim at reducing the dimension of the space depicted in Eq. (8). First, we can transform (8) into equalities as follows:

$$\begin{cases} k_d = M_\eta + M_1, \\ k_i = 0 + M_2, \\ k_p = k_d + \frac{2}{\alpha} k_i + M_3, \\ \alpha = \| \frac{-k_d}{(-k_d - 2M_\eta + \sum_{i=1}^{6} (\eta_i - \eta_{id}) \frac{\partial M_\eta}{\partial \eta_i})} \|_{max} + \epsilon, \end{cases} \tag{9}$$

where M_1, M_2 and M_3 are three 6×6 positive matrices and ϵ is a small positive constant. With this transformation (9), the fulfillment of the Lyapunov stability constraints (8) can now be verified by only assessing the value of $[M_1, M_2, M_3, \epsilon] \in \mathbb{R}^{3\times6\times6+1,>0}$. In order to further reduce this dimension space, we apply a diagonal transformation on the matrices M_i: $M_i = P\Lambda_i P^{-1}$, where $\Lambda_i \in \mathbb{R}^6$ are positive vectors and $P \in \mathbb{R}^{6\times6}$ is a positive invertible matrix chosen randomly beforehand. Thanks to this transformation, we can now assess the value of $[M_1, M_2, M_3, \epsilon] \in \mathbb{R}^{3\times6\times6+1,>0}$ (and K_p, K_i, K_d with Eq.9) by only accessing:

$$[\Lambda_1, \Lambda_2, \Lambda_3, \epsilon] \in \mathbb{R}^{19,>0}. \tag{10}$$

5. PID TUNING USING DEEP LEARNING

In order to take into account the process uncertainties, we propose to learn a stochastic predictive model π_ω (parameterized by ω) that maps the system state vector x into the controller parameters of the PID law:

$$\begin{cases} \pi: \quad \Omega_x \subset \mathbb{R}^{18} & \mapsto \Theta \subset \mathbb{R}^{109} \\ x = \left[p, \eta, \int_0^t e(\tau) d\tau \right]^T & \mapsto [k_p, k_i, k_d, \alpha] \end{cases} \tag{11}$$

This mapping is composed of two stages. The first stage is a trainable neural network whose input is the state vector x, aiming at predicting the vector $[\Lambda_1, \Lambda_2, \Lambda_3, \epsilon] \in \mathbb{R}^{19}$. To this end, we model each variable $n \in \{1, \ldots, 19\}$ of this vector as a random variable with an independent Normal distribution as:

$$\mathcal{N}_n(\mu_n, \sigma_n^2) = (2\pi\sigma_n^2)^{-1/2} \exp\{-\frac{1}{2\sigma_n^2}(x - \mu_n)^2\}, \tag{12}$$

The goal of this network is to output the 19 pairs of (μ, σ) representing the action distributions. We have empirically set up an architecture composed of 2 hidden layers of 32 hidden nodes each, and with the Sigmoid activation function applied to each layer. This results in a total of $(18 + 1) \times 32 + (32 + 1) \times 32 + 2 \times ((32 + 1) \times 19) = 2918$ parameters (ω) to learn from data. In a second stage of the mapping π, the PID parameters $[\Lambda_1, \Lambda_2, \Lambda_3, \epsilon]$ are obtained by sampling from the resulting Gaussian distributions $\mathcal{N}(\mu, \sigma)$. The Eq. 9 allows to compute the final PID parameters and using Eq. 5 the PID control inputs are derived. With this setting, we can more easily assess the Lyapunov stability of the system. The overall control strategy is illustrated in Figure 1.

6. LEARNING WITH THE CROSS ENTROPY METHOD

We propose to learn/optimize the weights of the neural network with the CEM. The CEM is a direct search method (i.e no gradient is computed). It is an Estimation of Distribution Algorithm (EDA) inspired by Natural Evolution Strategies Wierstra et al. (2008). With CEM, during one iteration k, N sets of weights $S^k_{i=1\ldots N}$ are sampled from a Normal distribution directly in the space of weights R^{2918}. At each iteration k, N evaluations are made to determine the current best weights $S^k_{best=1\ldots N\times\rho}$ with respect to a given cost function. In our case, we used the following classic control performance (based on multi steps within an episode and connected to the tracking error (Eq.2) index as a cost function to minimize:

$$J = \sum_{steps} \frac{1}{6} \sum_{i=1}^{6} (\eta_{d,i} - \eta_i)^2. \tag{13}$$

The mean and covariance of the Normal distribution are then updated as the mean and the covariance of the $N \times \rho$ best sets of weights obtained at iteration $k-1$. Noise σ^2_{noise} is added to the covariance of the best weights to avoid local optima. We randomly sample the next iteration weights $S^k = \mathcal{N}(\text{mean}(S^{k-1}_{best}), \text{Cov}(S^{k-1}_{best}) + \sigma^2_{noise})$.

7. EXPERIMENTS AND ANALYSIS

7.1 Experimental settings

Simulation: We control a simulated UUV in Gazebo as described in Section 3. The simulation is running in real-time and the frequency of the control loop is 20 Hz. The outputs of the ANNs, are used to compute the PID control law that is ultimately applied to the 6 thrusters. The complete training loop of the ANN is given in Figure 1.

Training setup: We used the following CEM hyperparameters that have been chosen through a grid search: population size $N = 25$, proportion to keep $\rho = 0.2$ and added noise $\sigma^2_{noise} = 0.1$. Each training episode is composed of 200 timesteps. One epoch is defined as performing one episode using each of the N sets of parameters candidate. The training consists in performing 200 epochs, thus a total of $200 \times 200 \times 25 = 10^6$ timesteps.

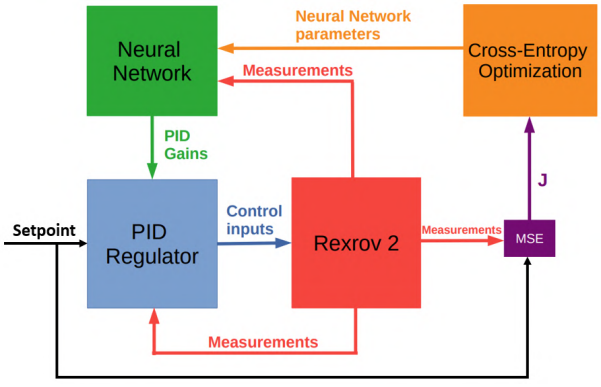


Fig. 1. Block diagram of the proposed method for PID tuning loop using Cross-Entropy Deep Learning.

Stability metrics: Thanks to the proposed Lyapunov function (6) and mapping (10), we can directly assess the state and parameters stability. We propose to measure the controller's stability as the respective percentage of steps satisfying the state and parameters constraints.

Evaluation setup: We propose three evaluation scenarios based on induced disturbance: none, Gaussian noise in sensor measurements, and Gaussian noise in control inputs with current disturbances. The length of the episode is now increased to 2000 timesteps. The initial state of the UUV is changed at the beginning of each episode. We compare our LB adaptive controller to its Lyapunov-based counterpart which consist in a PID controller whose parameters are set to: $M_1, M_2, M_3 = diag(0.5 - 10^{-5}, 0.5 - 10^{-5}, 0.5 - 10^{-5}, 0.5 - 10^{-5}, 0.5 - 10^{-5}, 0.5 - 10^{-5}) + 10^{-5} \times \boldsymbol{J}_{6\times 6}$ (which satisfies the parameters stability constraints in Eq.(9)). The resulting controller remains adaptive (as the gains are a function of the vehicle state $\mathbf{x}$) and will be denoted as naive PID henceforth. The only difference between these controllers is the value of the parameters used to derive the PID law (5), making the comparison fair.

7.2 Control performance

The control performance is measured as the MSE on the setpoint. The evolution of this performance is illustrated in the following figures: when facing no disturbance in figure 2; when facing Gaussian noise on the vehicle position and orientation feedback in figure 3 and when facing Gaussian noise on the control inputs and sea current disturbance in figure 4.

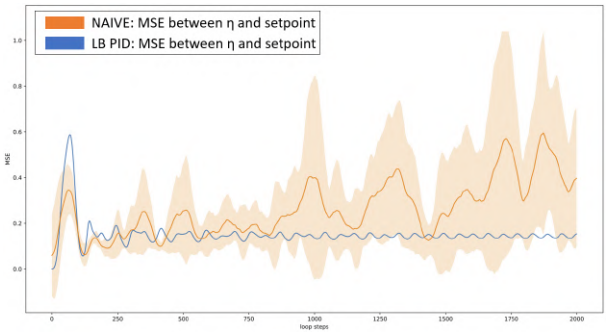


Fig. 2. Control performance without disturbance.

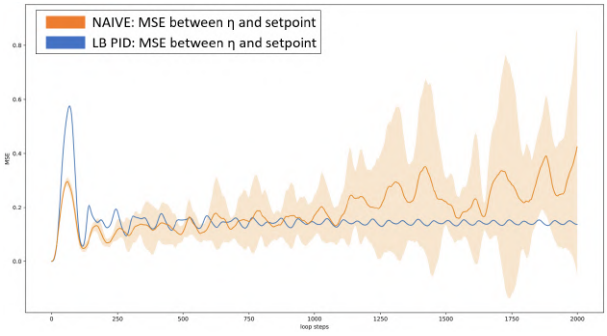


Fig. 3. Control performance with noisy position and orientation measurements.

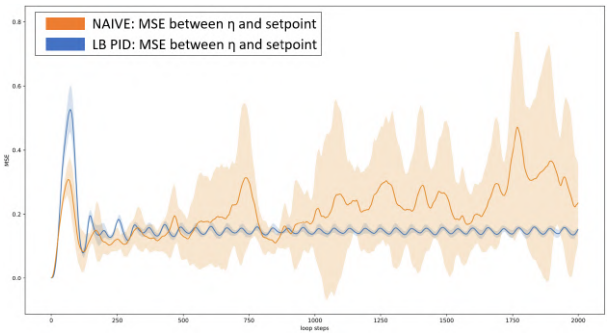


Fig. 4. Control performance with noisy control inputs and sea current disturbance.

We can see that the naive PID (in blue) results overall in better performances in terms of setpoint tracking compared to the LB PID (in orange). Nevertheless, we can see that the relative performance of the LB PID with respect to the naive PID's performance improves with increased uncertainty. In Figures 3 and 4, we can see that the performance of the LB PID matches the naive PID for approximately 500 timesteps, while without disturbance, its performance drops notably earlier as shown in figure 2. The evolution of the vehicle's state is represented in Figures 5-10. We can observe a difference in regulation dynamics depending on the DoF. For instance, we can see that the depth of the UUV is not successfully regulated by none of the controllers. Due to the short latency between episodes and the position of the UUV's CoG, the UUV slightly sinks at the beginning of the episode, altering its depth and yaw angle (z, yaw). This explains the vertical drift in their associated errors observed in the figures. In addition, as seen in Figures 5-10, the LB PID tends to regulate effectively more DoF compare to the naive PID (which fails at regulating (z, yaw), see Figures 5, 7, 9).

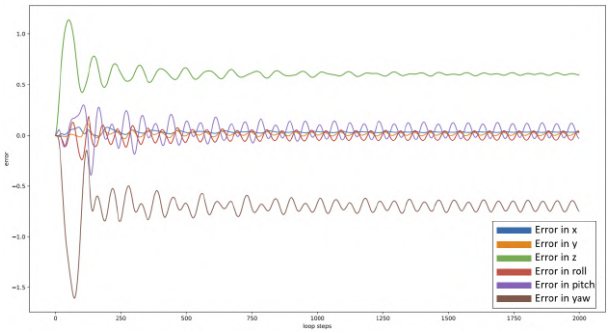


Fig. 5. The position and orientation errors of the naive PID during a non-disturbed episode.

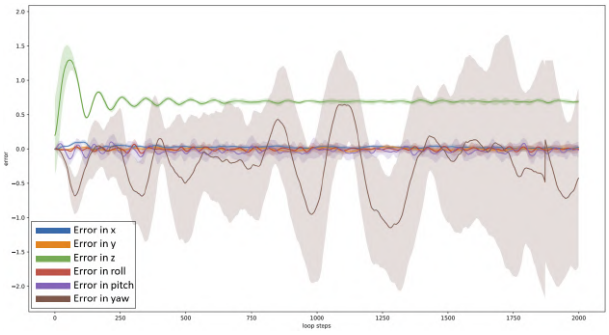


Fig. 6. The position and orientation errors of the LB PID during a non-disturbed episode.

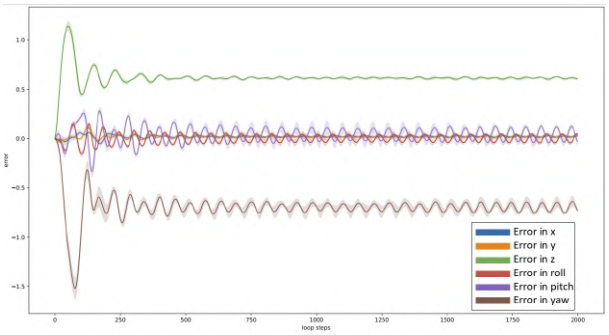


Fig. 7. The position and orientation errors of the naive PID during an episode with measurements noise.

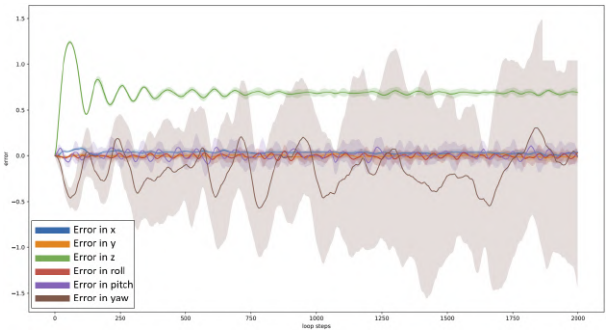


Fig. 8. The position and orientation errors of the LB PID during an episode with measurements noise.

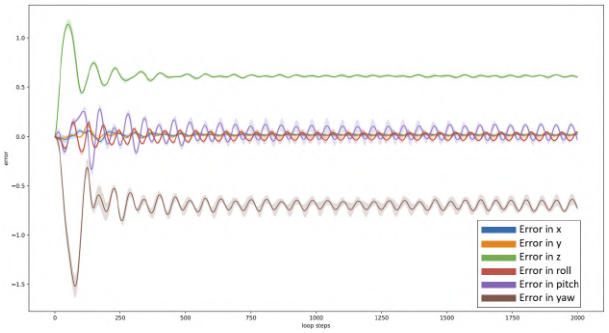


Fig. 9. Position and orientation errors of the naive PID with noisy control inputs and sea current disturbance.

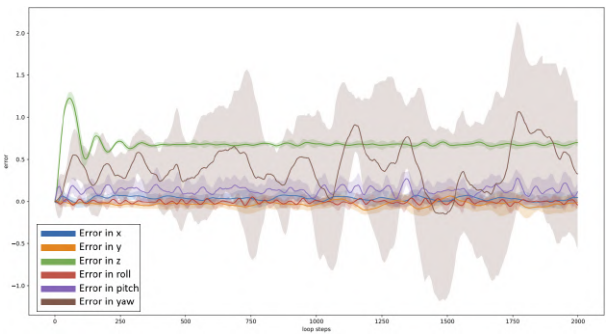


Fig. 10. The position and orientation errors of the LB PID with noisy control inputs and sea current disturbance.

7.3 Stability performance

The evolution of the stability metrics is illustrated in Figures 11-16. We can see in Figure 11 that both controllers reach a similar level of state stability. The naive PID exceeds the stability performance of the LB PID in both state and parameters stability. We believe that this is thanks to the stochastic nature of the adaptation (see Eq.(12)). Note that similar state stability does not guarantee similar control performances. Indeed, as mentioned before, both controllers reach the same level of state stability, however, we have seen that the MSE of the LB PID increases over time, so its control performance decreases but the control loop remains stable. Finally, we can see a mismatch between state and parameters stability: despite reaching a percentage of state stability as good as the naive PID, the gains obtained from the LB PID are not satisfying the constraints (8). This suggests that the Lyapunov function holds a limited conservatism. In other words, the Lyapunov-based space of parameters seems extremely small compared to the space of parameters obtained from the ANN. Future work could focus on incorporating the parameter constraints in the learning optimization scheme to analyze the difference in the resulting spaces.

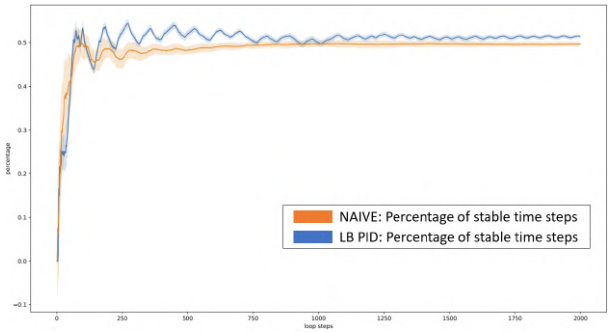


Fig. 11. Evolution of the state stability during a non-disturbed simulation

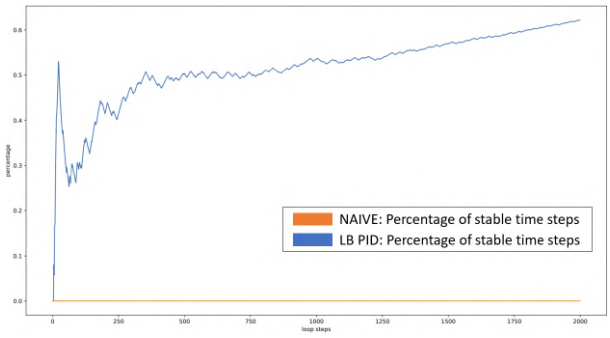


Fig. 12. Evolution of the parameters stability during a non-disturbed simulation

8. CONCLUSION

We proposed the use of CEM for PID tuning to adapt to process variation. Despite not considering stability components in the CEM procedure, the proposed LB PID is able to match the state stability obtained with the Lyapunov-based PID controller. Our result also suggests that the LB PID is better at compensating process variation. Learning-based adaptive control might be a key ingredient toward the safe deployment of fully autonomous UUVs.

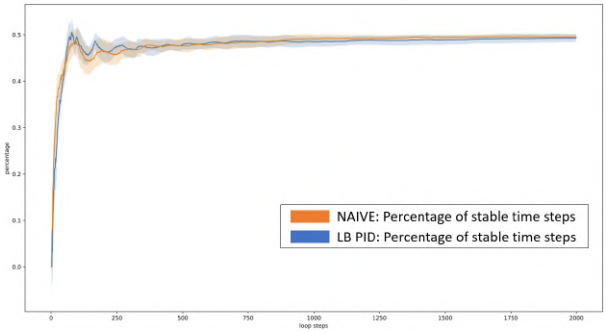


Fig. 13. Evolution of the state stability during a simulation with measurements noise

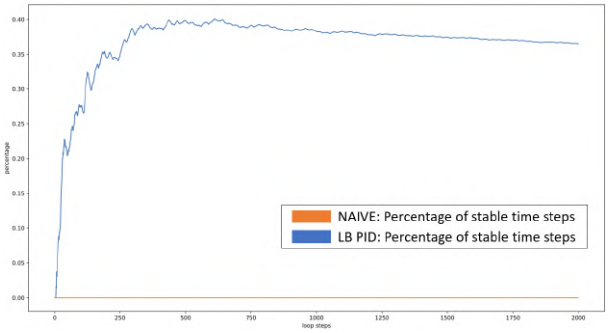


Fig. 14. Evolution of the parameters stability during a simulation with measurements noise

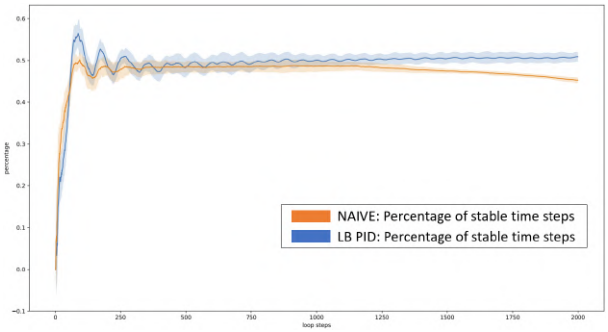


Fig. 15. Evolution of the state stability during a simulation with current and thrusters noise

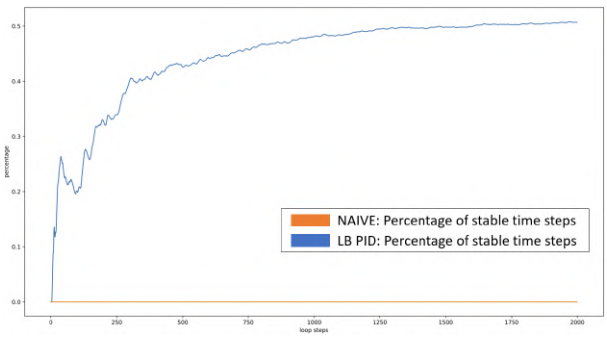


Fig. 16. Evolution simulation of the parameters stability during a simulation with current and thrusters noise

REFERENCES

Åström, K.J. and Wittenmark, B. (2013). Adaptive Control. Courier Corporation.

Berg, V. (2012). Development and Commissioning of a DP system for ROV SF 30k. Ph.D. thesis, NTNU.

Brunton, S.L. and Kutz, J.N. (2019). Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control. Cambridge University Press.

Chaffre, T., Chenadec, G.L., Sammut, K., Chauveau, E., and Clement, B. (2021). Direct adaptive pole-placement controller using deep reinforcement learning: Application to auv control. 13th IFAC CAMS.

Chaffre, T., Moras, J., Chan-Hon-Tong, A., Marzat, J., Sammut, K., Le Chenadec, G., and Clement, B. (2022). Learning-based vs model-free adaptive control of a mav under wind gust. Informatics in Control, Automation and Robotics, 362–385.

Fossen, T. (1994). Nonlinear Modelling And Control Of Underwater Vehicles. Ph.D. thesis, NUST.

Fossen, T.I. (2011). Handbook of Marine Craft Hydrodynamics and Motion Control. Wiley.

Jayachitra, A. and Vinodha, R. (2014). Genetic algorithm based pid controller tuning approach for continuous stirred tank reactor. Adv. Artif. Intell.

Knudsen, K.B., Nielsen, M.C., and Schjølberg, I. (2019). Deep learning for station keeping of auvs. In MTS/IEEE OCEANS.

Liberzon, D. (2005). Review of liapunov functions and stability in control theory. Automatica, 41.

Manhães, M., Scherer, S., Voss, M., Douat, L., and Rauschenbach, T. (2016). UUV simulator: A gazebo-based package for underwater intervention and multi-robot simulation. In MTS/IEEE OCEANS. Monterey.

McCue, L. (2016). Handbook of marine craft hydrodynamics and motion control [bookshelf]. IEEE Control Systems Magazine.

O'Dwyer, A. (2006). Handbook of PI and PID controller tuning rules. Imperial College Press.

Parks, P. (1981). Stability and convergence of adaptive controllers-continuous systems. In IEE Proceedings D-Control Theory and Applications. IET.

Rubinstein, R.Y. (1997). Optimization of computer simulation models with rare events. European Journal of Operational Research, 99, 89–112.

Sutton, R.S. and Barto, A.G. (2018). Reinforcement learning an introduction - Second edition. MIT Press.

Wang, C., Wei, L., Wang, Z., Song, M., and Mahmoudian, N. (2018). Reinforcement learning-based adaptive trajectory planning for auvs in under-ice environments. In MTS/IEEE OCEANS.

Wierstra, D., Schaul, T., Peters, J., and Schmidhuber, J. (2008). Natural evolution strategies. IEEE Congress on Evolutionary Computation.

Yang, R., Clement, B., Mansour, A., Li, M., and Wu, N. (2015). Modeling of a complex-shaped underwater vehicle for robust control scheme. Journal of Intelligent and Robotic Systems.

Learning Stochastic Adaptive Control using a Bio-Inspired Experience Replay

SUBMISSION DATE / POSTED DATE

03-03-2022 / 08-03-2022

CITATION

CHAFFRE, Thomas; SANTOS, paulo; LE CHENADEC, Gilles; CHAUVEAU, Estelle; CLEMENT, Benoit; Sammut, Karl (2022): Learning Stochastic Adaptive Control using a Bio-Inspired Experience Replay. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.19297577.v1

DOI

10.36227/techrxiv.19297577.v1

Learning Stochastic Adaptive Control using a Bio-Inspired Experience Replay

Thomas Chaffre, Paulo E. Santos, Gilles Le Chenadec,
Estelle Chauveau, Karl Sammut, *Senior Member, IEEE,* Benoit Clement

***Abstract*—Deep Reinforcement Learning (DRL) methods are dominating the field of adaptive control where they are used to adapt the controller response to disturbances. Nevertheless, the usage of these methods on physical platforms is still limited due to their data inefficiency and the performance drop when facing unseen process variations. This is particularly perceived in the Autonomous Underwater Vehicles (AUVs) context as studied here, where the process observability is limited. To be effective, DRL-based AUV control systems require the use of methods that are data-efficient (in order to reach a satisfactory behavior with a sufficiently fast response time) and are resilient (to ensure robustness to severe changes in operating conditions). With this ambition, we study in this paper the effect of the Experience Replay (ER) mechanism on the performance variation of a DRL-based stochastic adaptive controller. We propose a new ER method (denoted as BIER) that takes inspiration from the biological Replay Mechanism and compare it to the standard method denoted as CER. We apply it to the Soft Actor-Critic, a maximum entropy DRL algorithm, for use with an AUV maneuvering task that consists in stabilizing the vehicle at a given velocity and pose. The training results show that our BIER method exceeds the performance of the nonadaptive optimal model-based counterpart of the controller in less than half the number of episodes compared to CER. We proposed different evaluation scenarios of increasing complexity as measured by desired velocity value and amplitude of current disturbance. Our results suggest that the BIER method achieves improved learning stability and better generalization abilities.**

***Index Terms*—Deep reinforcement learning (DRL), neural networks, machine learning (ML), adaptive control.**

I. Introduction

A. Adaptive Control

Autopilots for unmanned systems are usually designed based on the feedback provided from velocity and orientation sensors. In the case of autopilot systems for autonomous underwater vehicles (AUVs), the main objective in the design is to compensate for waves and current-induced disturbing forces acting on their body. Existing AUV autopilots are however only able to compensate for low-frequency components of sea-induced disturbances. It seems natural to assume that the AUV performance could be improved by taking the nature of the disturbances into account in the design of the autopilot.

Adaptive control [1] provides what seems to be an ideal framework to this end. The objective of this technique is to adjust automatically the control parameters when facing unknown or time-varying processes such that the desired performance threshold is met. Developed in the late 1950s, adaptive control frameworks have been considerably expanded and used in various fields, their application has been facilitated by the rapid progress in microelectronics and the increasing interaction between laboratories and companies, from aerospace to maritime industries. As a result, adaptive controllers started to be widely adopted in the industry in the early 1980s. It was established at that time that *robust designs* with fixed parameters are too limited to handle complex regimes. The study of adaptive controllers for AUV manoeuvring is associated with various challenges, including:

Unknown dynamics: the uncertainty associated with describing precisely the states of waves or currents is high. This, together with its dynamic nature, prevents linear feedback control methods from achieving optimal performance of the plant. This becomes more critical in the presence of changes in weather conditions that impose a multiplicative factor in the component of the induced forces. The disturbance period will also vary with the speed of the vehicle and its orientation relative to the waves.

Nonlinearity: the controller response at some operating points must be overly conservative in order to satisfy the specification at other operating points. This is difficult to achieve for fixed parameters obtained through local linearization, that do not encompass the entire regime envelope.

Thruster efficiency: a fully-actuated vehicle can often become underactuated as its forward speed increases. This is especially true for hovering AUVs which are designed to move at low speeds and steer using their thrusters only. As the forward speed increases, the efficiency of lateral motions is drastically reduced, making it impossible for the platform to account for pure lateral motions.

System reliability: if the performances of one or more thrusters become increasingly less effective, the control system should be able to detect this and engage a new control algorithm specially designed to accommodate the failures and, if possible, to complete the mission.

A class of adaptive control methods, known as learning-based adaptive controllers, have been developed to tackle some of these limitations. This family of solutions uses machine learning algorithms capable of compensating for the unknown part of a process while also maintaining optimal control of its known part using traditional methods.

This work was supported by ISblue project, Interdisciplinary graduate school for the blue planet (ANR-17-EURE-0015) and co-funded by a grant from the French government under the program "Investissements d'Avenir".

The first author's scholarship is jointly supported by the doctoral school of Bretagne-Loire University with a grant from the Brittany Region under the program "Mobilité Internationale" and by Flinders University with a grant provided by the South Australian Government.

B. Learning-based adaptive control

Real-world systems are in general nonlinear and their motion equations, parameters and system measurements are affected by uncertainty. A realistic scheme is to consider that the process model is partially available. In learning-based adaptive controllers, model-free algorithms are used to mitigate this lack of a complete description of the process by finding (*learning*) an approximate representation of the unknown parts, or by fitting (*tuning*) the best control parameters for a target behavior. The dynamics of such a system can be represented as the sum of known (f_1) and unknown (f_2) parts:

$$\begin{aligned} \dot{x}(t) &= f_1(t, x, u) + f_2(t, x, p), \\ y(t) &= h(t, x, u), \end{aligned} \tag{1}$$

where classical model-based control methods can be used to efficiently control f_1, and f_2 which can be approximated by model-free learning algorithms. In other words, learning-based control methods take advantage of the fast convergence and robustness to uncertainty of learning algorithms to approximate an unknown performance function, while applying model-based control laws to maintain some stability requirements of the system. More recently, in Neural-Network (NN) learning-based control design, the unknown part of the model can be estimated by a NN, whose weights are obtained using some model-free optimization procedure. Among the various techniques, a prominent candidate to that end is Deep Reinforcement Learning (DRL).

In DRL, the learning process happens through the interaction between an agent and the environment [2]. DRL is traditionally defined as a Markov Decision Process (MDP) expressed as the tuple $\langle S, A, T, R \rangle$, in which:

- S is the set of possible states;
- A is the set of actions that can be executed by the agent;
- T is the transition function that defines the probability of reaching a successor state $s' \in S$, from the application of an action $a \in A$ in a state $s \in S$;
- R is the reward function.

The DRL general framework can be summarised as follows:

1) At an instant t, an action $a \in A$ is chosen by the agent in a state $s \in S$;
2) The execution of this action leads the agent to a state $s_{t+1} \in S$ and the agent receives a scalar value r_t, the reward signal. This signal is a numerical representation of the action outcomes with respect to a reward function $R(s)$. The goal of DRL is to maximize this reward function;
3) Finally, the agent updates the value of executing action a based on the received reward, according to the specific DRL algorithm applied.

In the domain of optimal control, the *agent* is identified with the *controller*, *environment* is the *controlled system* (or *plant*) and *action* is the *control signal* [2].

Among the various existing DRL algorithms [3], Deep Policy Gradient methods (that use gradient descent for the purpose of optimizing a decision making function, denoted as policy, with respect to the expected return) are deemed the most suitable method for handling robotic domains for the following reasons:

1) Domain dimensionality: the large size of the state-space in a real robot environment makes the application of value iteration based methods unfeasible;

2) Non-observable disturbances: any real environment, beyond toy scenarios, is partially observable. In this context, environment modelling and trajectory planning are extremely difficult tasks for any model-based or value-based DRL method, since unexpected states can corrupt the remaining part of the estimated trajectory, resulting in failures.

Deep Policy Gradient methods are essentially relying on the Actor-Critic architecture [4], where a value and a policy function are estimated simultaneously in order to improve the agent performance. This formulation has led to the development of what is known as off-policy methods where the estimates of these functions can be improved using the Experience Replay (ER) mechanism. ER is the state-of-the-art method for the automatic selection of past experience to improve an agent's future behavior, allowing the use of samples from various distribution. In contrast, with the on-policy formulation, only samples generated by the same policy can be considered for the optimization process.

The off-policy Deep Policy Gradient methods are increasingly being applied to AUVs adaptive control [5]–[8] where the standard ER method is used to improve DRL-based agents for the purpose of adapting to process variations. These works rely on deterministic policies that are highly stable during training but display lower adaptation abilities compared to their stochastic counterparts. Nevertheless, the performance of Deep Policy Gradient methods is highly sensitive to the distribution shift problem that is in the context of DRL, the difference between the training and evaluation set of states. A key element toward the reduction of this sensibility is the ER mechanism, and its effect on the resulting policy is the main focus of this paper.

This paper is organized as follows: in Section II the ER mechanism is introduced with its standard formulation in DRL and its biological counterpart. In Section III we present an application of learning-based adaptive control theory for the control of an AUV. The detailed architecture of the learning-based adaptation is provided in Section III-B. We present our proposed ER approach in Section IV. The training settings are provided in Section V along with the training results. The evaluation scenarios are described in Section VI with an analysis of the results, leading to some open questions and perspectives for future work.

II. Experience Replay

The concept of ER [9] employs the agent's past experience to improve its current behavior. It aims to artificially make the agent's experience look Independent and Identically Distributed (IID). This is highly desirable in order to not concentrate the updates to a limited area of the desired functions.

Given that an agent's experience at the time step t is defined as the quadruplet $e_t = (s_t, a_t, r_t, s_{t+1})$, the ER method

consists of storing (at each time step) the experience e_t in a memory unit $\mathcal{D} = \{e_1, \ldots, e_t\}$ of fixed size, also known as the *replay buffer*. Then, the neural networks are trained by performing mini-batch gradient descent of past experiences randomly pooled over the replay buffer. The estimators are hence trained on IID samples that are generated by various trajectories and policies and they are, therefore, more representative of the true function. However, even with this first ER formulation, there is a great number of parameters that are usually ignored despite having an impact on the learning performance, including:

The replay buffer size: the total number of transitions that the replay buffer can store. When its maximum size is reached, the replay buffer is accessed in a first-in-first-out fashion. The bigger the replay buffer size, the more the data will look like it is IID, which in turn improves the gradient update quality. However, if the replay buffer is too big, an important transition will have much less chance of being used to update the policy, which could impair the learning process. In contrast, if the replay buffer is too small, the learned policy can be the result of an overfitting process on recent transitions, which precludes performance improvement.

The age of a transition: the number of gradient steps taken by the agent since the transition was generated. This value can be seen as a measure of the extent to which the transitions stored in the Replay Buffer are off-policy, as it tells us how different the current policies are from those stored in the buffer. The age of the oldest policy stored increases with respect to the buffer size.

The replay ratio: the number of gradient updates per environment transition. It can be viewed as a measure of the frequency at which the agent is learning using existing data versus learning from collecting new experiences.

The size of the replay buffers, however, can impact negatively on the learning performance [10]. There are two competing methods that can be used to solve this issue: the Combined ER (CER) [10] and the Prioritized ER (PER) [11]. CER consists of adding the latest transition performed to the mini-batch pooled over the replay buffer, whereas with PER important transitions, as measured by their associated TD-error (22)(23), are given a higher probability to be used in the gradient updates. Using CER, however, the last transition will undoubtedly be sampled and instantly affect the policy.

Nevertheless, even with CER, a drop in performance was observed for certain sizes of replay buffer, at some point of the training (even when tuning the learning rate). This behavior was related to the process itself rather than to the aforementioned parameters [10]. As written in [10] "CER is a workaround ... and future effort should focus on developing a new principled algorithm to fully replace ER." In this paper, we propose a new ER mechanism with the ambition to decouple the performance of the agent from process complexity (as observed with CER).

A recent detailed analysis of ER was provided in [12], where an analysis of the effects of the aforementioned parameters was presented. Several conclusions on how the parameters can affect the learning dynamics were drawn, which motivated the ER design proposed in this work. These conclusions can be summarised as follows:

- Increasing the replay capacity while fixing the age of the oldest policy improves the performance because it lowers the chances of overfitting to a small subset of (state,actions).
- As the agent trains, it spends more time in higher quality regions of the environment (as measured by rewards), thus learning to better estimate the return in such regions leads to further gains in performance.
- Increasing the buffer size with a fixed replay ratio has varying improvements. The replay ratio stays constant when the buffer size is increased because of both the replay capacity and the age of the oldest policy increase. If one of these two factors is independently modulated, the replay ratio will change.

The design of the proposed approach for ER has been mostly motivated by these findings which we tried to incorporate in the ER scheme. A recent study [13] undertook a comprehensive comparison between the replay mechanism that takes place in biological brains and those in artificial learning systems. We list below some findings related to DRL only.

Replay (in biological systems) is temporally structured. Temporally correlated experience sequences are used for learning and memory combination. This allows for more combinations of neurons which leads to faster emergence of temporal waking experiences. This feature is largely ignored by existing methods that only replay static, uncorrelated inputs.

Replay is modulated by reward and only a few selected experiences are replayed. It seems intuitive that not all experiences are useful for learning a new task. Some experiences are more important than others because they incorporate higher quality information about the process dynamics. The challenge here is twofold: how to model this information quality and how to measure it.

Replay is treated differently for novel versus non-novel inputs. This allows for selective replay to be weighted by novelty. Biological systems tend to reduce drastically the attention given to old experiences versus that given to recent ones as the latter contain more information within them.

In this paper, we propose a new ER mechanism to include these biological insights while keeping in mind the constraints related to the regression problem.

III. A LEARNING-BASED ADAPTIVE CONTROLLER

In this paper, we address the control problem of AUV manoeuvring, which can be summarized as the stabilization of an underwater vehicle at a fixed velocity and orientation (x_{ref}, with $\dot{x}_{ref} = 0$). The state vector is hence defined as $x = [x\ y\ z\ \phi\ \theta\ \psi]^T$. The vehicle is fully actuated but subject to external disturbances (sea currents) that are here not observable (noted that relative current can be estimated with some types of DVL incorporating a bottom lock). As defined in Section I-B, the process dynamics can be framed as the combination of its known f_1 and unknown f_2 parts. Our approach consists in using the knowledge of f_1 to design a model-based control structure, which is then combined

with a model-free learning algorithm to compensate for f_2. The resulting learning-based controller guides the controller parameters toward the optimal ones that best compensate for both f_1 and f_2 without prior estimation of some process parameters, leading to a Direct Adaptive Control scheme. Let the error between the present ($\bar{x}_i$) and the desired (x_{ref_i}) state variable be defined as $e_i = x_{ref_i} - \bar{x}_i$.

The task of steering the AUV outputs in order to maintain the error signals within a specific threshold, over a predefined amount of time (guaranteeing the vehicle stabilization), can be achieved if the following control objective is met:

$$\forall i \in \mathbb{R}^u,\ \forall\ t' \in [t-\varsigma, t],\ \exists!\ \mid e_i(t') \mid > \chi, \tag{2}$$

where $\mathbb{R}^u$ is the space of control inputs, the current time step is denoted as t and ς is the length of the period of time over which we want all the errors e_i to be less than the desired value χ. This class of control objective is used in various AUV missions, such as autonomous docking or underwater inspection, where a conservative regulation of the vehicle's outputs is required.

We use the ROS-based UUV Simulator [14] which contains a small library of different vehicles of cubic and torpedo shapes. For this study, we used the RexROV2 platform, a cubic shape AUV[1]. The UUV Simulator can simulate several current and wave disturbances, thruster dynamics, and body wrench disturbances. When incorporated in simulations, the induced forces have a realistic physical impact on the robot and fluid dynamics. The sea current disturbance (which is the main focus of this study) is modeled as a uniform force acting over the Gazebo environment. This force is represented by a linear velocity, v_c (in $m.s^{-1}$), a horizontal h_c and a vertical angle j_c (measured in radians). The simulated RexROV2 platform is equipped with an IMU which feedbacks its linear velocities and orientation (Euler angles). These variables are accessible through ROS topics, which are essentially data pipelines to access the state of the simulated objects. Our software architecture consists in using the simulation metadata to train the learning algorithms considered in this work.

A. Design of the model-based part of the controller

We introduce now a simple learning-based adaptive controller formulation. We first proposed it in [15] where we demonstrated its superiority to its nonadaptive optimal model-based counterpart (denoted in this manuscript as OFP for Optimal Fixed Poles controller) that we used here to compare our controllers. In particular, we show in [15] that only the learning-based adaptive controller (that we introduce now) was able to regulate the vehicle against time-varying disturbances. Moreover, we provide in this paper an enhanced model-based design that leads to better performance. This work assumes that the controlled vehicle is fully observable and controllable. This means that each of the vehicle's DoF is measurable and the desired vehicle states (within the operating regimes) are supposed to be accessible. In this context, a PID controller is a suitable method to regulate the process. We can take into account the current disturbance by considering the steady-state error variable $\sigma = \int_0^t e(\tau)d\tau$. We can rewrite the state-space equations with the augmented state vector $X = [\sigma, e, \dot{x}]$ as:

$$\frac{d}{dt}\begin{bmatrix}\sigma\\ e\\ \dot{x}\end{bmatrix} = \underbrace{\begin{bmatrix}0&1&0\\0&0&1\\0&0&0\end{bmatrix}}_{\text{A}}\begin{bmatrix}\sigma\\ e\\ \dot{x}\end{bmatrix} + \underbrace{\begin{bmatrix}0\\0\\1\end{bmatrix}}_{\text{B}} u. \tag{3}$$

The PID state-space representation is given by:

$$\dot{X} = (A - BK)X. \tag{4}$$

The PID control law can then be derived as:

$$u = k_p e + k_i \sigma + k_d \dot{x}, \tag{5}$$

with k_p, k_i and $k_d \in \mathbb{R}^+$. Among the various procedures and rules that can be applied to tune the PID gains [16], a fundamental technique consists of assigning a set of specific values, $P = \{\lambda_1 \lambda_2 \dots \lambda_n\}$, to the eigenvalues of the feedback loop $A-BK$. Given that these eigenvalues determine the poles of all the transmittances where the associated state matrices are involved, this procedure is denoted as Pole-Placement. We can define a (normalized) control polynomial as

$$C(s) = s^n + c_1 s^{n-1} + \dots + c_{n-1}s + c_n, \tag{6}$$

whose roots are the λ_i, which can be assigned the characteristic polynomial of $A - BK$ with:

$$C(s) = det(s\mathbf{I} - (A - BK)). \tag{7}$$

Equations (4) and (7) yield

$$\begin{aligned} |A - BK - \lambda\mathbf{I}| &= -\lambda(\lambda(k_d + \lambda) + k_p) - k_i, \\ &= -\lambda^3 - \lambda^2 k_d - \lambda k_p - k_i, \\ &= 0. \end{aligned} \tag{8}$$

The desired λ_i are solutions of

$$\lambda^3 + \lambda^2 k_d + \lambda k_p + k_i = 0. \tag{9}$$

To ensure the control loop stability, the poles of (9) must be placed in the complex left half-plane (i.e. $A-BK$ is Hurwitz). For this purpose, the poles of the controller (5) must be solutions to (9). In order to derive such poles $(\tau_1, \tau_2, \tau_3) \in \mathbb{R}^+$, we the following eigenvalues design is considered:

$$\lambda_1 = \frac{-1}{\tau_1};\ \lambda_2 = \frac{-1}{\tau_2};\ \lambda_3 = \frac{-1}{\tau_3} \tag{10}$$

The Pole-Placement design can be written as:

$$\begin{cases} \frac{-1}{\tau_1^3} + \frac{k_d}{\tau_1^2} - \frac{k_p}{\tau_1} + k_i = 0 \\ \frac{-1}{\tau_2^3} + \frac{k_d}{\tau_2^2} - \frac{k_p}{\tau_2} + k_i = 0 \\ \frac{-1}{\tau_3^3} + \frac{k_d}{\tau_3^2} - \frac{k_p}{\tau_3} + k_i = 0 \end{cases} \tag{11}$$

Since τ_1, τ_2 and τ_3 are solutions to (9), it follows that:

$$\begin{bmatrix}1&\frac{-1}{\tau_1}&\frac{1}{\tau_1^2}\\1&\frac{-1}{\tau_2}&\frac{1}{\tau_2^2}\\1&\frac{-1}{\tau_3}&\frac{1}{\tau_3^2}\end{bmatrix}\begin{bmatrix}k_i\\k_p\\k_d\end{bmatrix} = \begin{bmatrix}\frac{1}{\tau_1^3}\\\frac{1}{\tau_2^3}\\\frac{1}{\tau_3^3}\end{bmatrix} \Leftrightarrow MK^T = N \tag{12}$$

[1]See our last study [15] for further information on the simulated vehicle.

The resulting gains of the control law (5) are obtained by transforming back the poles with $K^T = M^{-1}N$ as:

$$k_i = \frac{1}{\tau_1\tau_2\tau_3}; k_p = \frac{\tau_1+\tau_2+\tau_3}{\tau_1\tau_2\tau_3}; k_d = \frac{\tau_1\tau_2+\tau_1\tau_3+\tau_2\tau_3}{\tau_1\tau_2\tau_3} \tag{13}$$

With the mapping in Eq. (13), the bounds for the controller parameters can be defined based on control constraints that are easier to derive in the poles domain. In the present case, with the design of Eq. (10), for any $\tau_i > 0$, the poles of the feedback loop are placed on the x-axis of the complex left half-plane. By doing that, we set the desired oscillation frequency and percentage overshoot to 0. This leaves us with the settling time requirement to define in order to set the desired distance from the y-axis and thus bound the value of the space of poles. In accordance with the control objective shown in Eq. (2), we define the desired maximum settling time of the closed loop control $\varsigma = 10$ seconds as the maximum time after which we want the system outputs to stay around $\chi = 5\%$ of its desired values. The upper bound of the space of poles is derived as:

$$\lambda_{max} = \frac{\ln(\chi)}{\varsigma} = \frac{\ln(0.05)}{10} = -0.3. \tag{14}$$

Therefore $\lambda_i \leq \lambda_{max}$ (Eq. (10)), from which we can derive the upper bound of the poles

$$\tau_{min} < \tau_i \leq \frac{1}{-\lambda_{max}} = 3.333 = \tau_{max} \tag{15}$$

We set $\tau_{min} = 0.025$ because, for lower values, the control inputs are too expensive in terms of control efforts and too aggressive for our control objective. Thus, the bounds of the poles are defined as:

$$0.025 \leq \tau_i \leq 3.333 \tag{16}$$

There is a solution for all $C(s)$ in Eq. (6) if and only if the pair (A,B) is controllable, that is assumed here. In the case of a Single-Input system, the solution is unique. In the case of Multi-Input systems, as studied here, the number of free components of the matrix K is greater than the n eigenvalue constraints. Accordingly, there exist multiple solutions, among which it is not trivial to define an optimal one.

When no model of the uncertainties is provided, as investigated here, model-free adaptation can be exploited. In order to take into account the uncertainties in the poles selection, we apply DRL to build a stochastic predictive model π_μ that maps a state vector s_t into the pole values:

$$\begin{aligned} &\pi_\mu : \ s_t \mapsto [\tau_{vx} \ ; \ \tau_{vy} \ ; \ \tau_{vz} \ ; \ \tau_\phi \ ; \ \tau_\theta \ ; \ \tau_\psi \ ], \\ &\pi_\mu : \ s_t \mapsto \mathcal{N}(\tau_i), \end{aligned} \tag{17}$$

where $dim(\tau) = 18$ and $\mathcal{N}(\tau_i)$ is the probability density function of τ_i that is modeled by a Normal distribution as:

$$\mathcal{N}(\tau_i) = (2\pi\mu_i)^{-1/2} \exp\Big\{-\frac{1}{2\mu_i}(x-\lambda_i)^2\Big\}, \tag{18}$$

where $\lambda_i \in \mathbb{R}$ and $\mu_i \in \mathbb{R}^+$ are the mean and variance of $p(\tau_i)$ that are estimated by the Policy network. Therefore, the outputs of the Policy network are the 18 pairs of (λ, μ) representing the Normal distributions $\mathcal{N}(\tau_i)$ used to sample the poles for each degree of freedom. Designing this stochastic function (17) is numerically expensive due to the dimensions of the underlying spaces, excluding real-time computation with model-based methods only. The DRL framework allows us to iteratively build an estimate of this optimal mapping function. We compare the resulting DRL-based controllers to the OFP controller (provided by the UUV Simulator) whose pole values have been optimized using SMAC [17], a model-based method, thus without learning. The OFP controller is not adaptive as the poles are fixed. Details on the resulting poles can be found on the UUV Simulator website[2]. We present in the next section our use of DRL to adapt the pole values.

B. Design of the DRL-based model-free learning procedure

The related methods cited earlier in Section I are mostly relying on the DDPG and TD3 algorithms. They are known to involve intensive tuning of hyperparameters to work properly. Additionally, in order to take into account the ocean current disturbances, one designer might favor a stochastic policy that is known to be more robust to uncertainties and to partially observable processes (at the cost of being less stable during training). For these reasons, we chose to use another Deep Policy Gradient method, named Soft Actor-Critic. It has been exploited to solve many DRL benchmark environment using the exact same NN architecture and hyperparameters. A complete description of this algorithm is detailed next. Our learning-based architecture is therefore composed of 3 main items that we present now.

1) Soft Actor-Critic: The Soft Actor-Critic (SAC) [18]–[20] is a current state-of-the-art Policy Gradient algorithm integrating three key components:

- An improved exploration and stability in performance thanks to entropy maximization [21].
- An Actor-Critic architecture [4] with separate Values and Policy networks.
- An off-policy formulation enabling the use of past collected data with Experience Replay [9].

Instead of optimizing only the expected sum of rewards, the objective function of SAC also maximizes the entropy of the behavior policy, that is weighted by a constant, α, as follows:

$$J_{SAC}(\pi_\mu) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\mu}}[r(s_t,a_t) + \alpha H(\pi_\mu(.|s_t))], \tag{19}$$

where

$$H(\pi_\mu(.|s)) = -\sum_{a\in A} \pi_\mu(a|s)\log\pi_\mu(a|s) \tag{20}$$

is known as the Shannon entropy measure of the policy π that is represented by an ANN parameterized by μ. By trying to maximize the entropy of the policy and the reward at the same time, the policy is driven to take the best actions while remaining as random as possible. This results in better exploration and improved robustness to uncertainty thanks to entropy maximization [21], [22].

[2]https://uuvsimulator.github.io

For this purpose, the entropy term is explicitly incorporated in the State-Value function $V(s_t)$ as:

$$\begin{aligned} V(s_t) &= \mathbb{E}[Q(s_t, a_t) + \alpha \mathcal{H}(\pi_\mu(.|s_t))], \\ &= \mathbb{E}[Q(s_t, a_t) - \alpha \log \pi_\mu(a_t|s_t)], \end{aligned} \tag{21}$$

where α is controlled indirectly by the reward scale (see Section V-B). In order to reduce Actor-Critic value overestimation [23], [24], the State-Value function is estimated by an ANN parameterized by Ψ using the minimum of two different Q-Value estimates represented by two ANNs parameterized by Υ_1 and Υ_2, respectively. We used TD-Learning [25] to iteratively build an estimate of the State-Value function (22) and Q-Value function (23). For the State-Value function, the parameters Ψ are thus optimized to minimize the TD-error:

$$\begin{aligned} J_V(\Psi) = V_\Psi^{\pi_\mu}(s_t) - \Big(&\min \left[Q_{\Upsilon_1}^{\pi_\mu}(s_t, a_t), Q_{\Upsilon_2}^{\pi_\mu}(s_t, a_t) \right] \\ &- \log \pi_\mu(.|s_t) \Big). \end{aligned} \tag{22}$$

Similarly, the parameters Υ_i of the i-th Q-Value function estimator are optimized to minimize the TD-error:

$$J_Q(\Upsilon_i) = Q_{\Upsilon_i}^{\pi_\mu}(s_t, a_t) - \big(r(s_t, a_t) + \gamma \times V_{\Psi'}^{\pi_\mu}(s_{t+1}) \big), \tag{23}$$

where $\gamma = 0.99$ is the discount actor and Ψ' is defined in Section III-B2. The parameters μ of the Policy network are then optimized [18] in order to minimize the expected Kullback-Leibler (KL) divergence between the current policy and the exponential of the Q-Value function that is normalized by a function Z_Υ as:

$$J_\pi(\mu) = \mathbb{E}_{s_t \sim D} \Big[D_{KL} \big(\pi_\mu(\cdot|s_t) || \frac{Q^*(s_t, \cdot)}{Z_\Upsilon(s_t)} \big) \Big], \tag{24}$$

where,

$$Q^*(s_t, a_t) = \exp \left(\min \left[Q_{\Upsilon_1}^{\pi_\mu}(s_t, a_t), Q_{\Upsilon_2}^{\pi_\mu}(s_t, a_t) \right] \right), \ \forall s_t, a_t. \tag{25}$$

When using the distribution expressed in Eq. (25) as a target for the policy shown in Eq. (24), the agent is forced to explore actions proportionally to their associated exponential Q-Values. This positive transformation allows a smarter exploration-exploitation tradeoff as negative Q-Values will be transformed into small but positive ones, forcing the policy to make progress along sub-optimal strategies until the algorithm finds which value is better for the future gains.

The gradient of $\nabla_\mu J_\pi(\mu)$ is approximated in [18] by:

$$\begin{aligned} \hat{\nabla}_\mu J_\pi(\mu) = \nabla_\mu \log \pi_\mu(a_t|s_t) + \Big(&\nabla_{a_t} \log \pi_\mu(a_t|s_t) \\ - \nabla_{a_t} \min \big(Q_{\Upsilon_1}^{\pi_\mu}(s_t, a_t), &Q_{\Upsilon_2}^{\pi_\mu}(s_t, a_t) \big) \nabla_\mu f_\mu(\epsilon_t, s_t) \Big). \end{aligned} \tag{26}$$

The derivative in Eq. (26) is easier to compute compared to that in Eq. (24) and allow the use of Gradient Descent to optimize the parameters μ of the Policy ANN. Considering Eq. (21), the parameters μ are consequently optimized exactly for the desired maximum entropy objective (19). The soft Q-update (23) guarantees that $Q^{\pi_{new}}(s_t, a_t) \geq Q^{\pi_{old}}(s_t, a_t)$ and the repeated policy updates (26) ensure convergence toward the optimal policy π^*[3].

[3]See Appendix B.2 and B.3 of [18] for the mathematical proof).

2) Stabilizing TD-Learning: The TD learning algorithm, summarized in Eq. (22) and (23), is different from Gradient Descent in the sense that the target value changes at each update, transforming the loss landscape. This generates training behaviors that are unstable and divergent. To tackle this problem, it is common practice to have a separate copy of the considered value network, denoted as "target" network, whose parameters are moving slowly or are fixed over a predefined amount of iteration steps. Here, we define a Target State-Value network $V_{\Psi'}(s)$ that is then used to compute the Q-Value TD error (Eq. (23)) and thus with Ψ' slowly tracking the value of Ψ. In our case, we used the soft updates procedure which, for the target network parameters, consists in slowly tracking the parameter of the State-Value function using an exponential moving average $\Delta = 0.005$. We used the Smooth L1 loss function from Pytorch [26] for the Critics optimization as it is less sensitive to outliers compared to standard MSE.

3) Artificial neural networks: Our implementation of the SAC algorithm is composed of 5 fully-connected Multilayer Perceptron (MLP): two Q-Value networks (with shared architectures), a Value and a Target-Value networks (with shared architectures) and a Policy network. We used the same ANN architecture as proposed in the original SAC paper [18] where each network is composed of 2 hidden layers of 256 hidden units each. The Pytorch framework [26] and CUDA toolkit [27] were used to implement this architecture along with an Nvidia RTX 2070 GPU card for the gradient and simulation processing. The SAC algorithm builds a stochastic policy where the action distributions are modeled by Gaussian distributions[4]. There are several advantages of considering a stochastic policy: it prevents early convergence of the policy variance, it encourages exploration in the value function by increasing the value of regions of state space that lead to high-entropy policy, and the resulting policy tends to perform much more consistently compared to its deterministic counterpart with improved robustness to uncertainties [18]. We used Adam [28] as optimizer for all networks with the learning rate $l_r = 3e^{-4}$. The Leaky ReLU activation function is applied to all hidden layers and gradient descent is applied using a mini-batch of size 256. Layer Normalization [29] (LN) is added before the activation function of all the hidden layers and L2 weight penalty of $1e^{-3}$ is incorporated to the Critics only. The weights and biases are initialized from the Gaussian distribution $\mathcal{N}(0, \sqrt{2/f})$, where f is the fan-in of the layer.

IV. A Bio-inspired Experience Replay

This section presents the ER method proposed in this paper, which is called BIER for Bio-Inspired ER. In this method, the agent experience is divided into two distinct memory units (as illustrated in Figure 1) and samples are drawn from them differently as described next.

Sequential-Partial Memory: this buffer is denoted as B_1 and is similar to the one from the original ER scheme. Here its maximum size is set to $1,000,000$, and contains old and new transitions. We believe that, especially in the robotic case,

[4]See appendix C of [18] for more details on the change of variable applied to bound the Gaussian distributions

the optimal behavior is highly temporally correlated. With robots, early actions do have an impact on the future states. Learning this temporal relationship is thus essential in order to learn what a truly good behavior, in the physical sense, is. In addition, even with very different operating conditions, the robot's behavior remains quite similar. This means that the shape of the trajectories remains within a bounded space. Therefore, our hypothesis is that learning on sequences can lead to further gains (compared to temporally uncorrelated samples) because what we learn on a trajectory can directly be applied to future ones that the agent has yet to encounter. Following this intuition, we propose to sample random sequences of transition from Buffer B1 (i.e. transitions that are successive within the buffer since we store each of them in a *first-in-first-out* fashion). The vector of experiences E sampled from B_1 is composed of n samples as $E = [e_i \ ; \ e_{i+1} \ ; \ \dots \ ; \ e_{i+n}]$, for $0 < i < m - n$.

While this replay procedure sounds very appropriate for a biological system, here we are optimizing MLPs with limited numbers of parameters and learning abilities. Using highly correlated samples when performing gradient descent often leads to overfitting or local minima. In fact, sequential states are strongly correlated. This is often observed in on-policy methods where the ANNs are optimized using samples generated by the current policy. This usually leads to repeatedly overfitting to those local correlated samples and never really learning the true value of the functions (i.e. it ends up oscillating between different overfitting regimes). For this reason, we propose to consider sequences that are only partial by storing 1 out of 2 transitions in the buffer B_1 which:

- adds a regularization effect by feeding incomplete sequences to the ANNs that encourages these networks to further learn the real value of the functions.
- reduces the age of the oldest policy contained in this buffer, which improved performance in [12].

Optimistic Memory: both bad and good behaviors are important when learning a new task because they both contain information about the process. We have been able to observe a number of cases where "positive reinforcing" is much more efficient with biological systems, for example with animal training where good and bad behaviors are respectively rewarded with treats or no response (rather than punishment). It was shown in [12] that trying to estimate values of high-quality regions (as measured by the rewards) results in better performance. In addition, as the agent learns, its performance improves and better transitions are performed. Such transitions are important because they can further improve the data collection of the agent in the future. However, as shown in [10], when using a large replay buffer, such transitions are likely to influence the policy later. Their probability to be sampled decreases as the replay buffer size increases, slowing down performance improvement. Following these heuristics, our objective with this second buffer B_2 is to be optimistic about past experience, by increasing the probability of using transitions associated with such high-quality regions.

We propose to store in B_2 the upper outliers of the reward distribution that we consider to be the best transitions. Outliers can be defined according to diverse metrics depending on the nature of the variable distribution. The challenge is that we can not predict the distribution shape beforehand. For instance, with our reward function, the closer the robot gets to the setpoint, the higher the maximum value of possible reward becomes (hence, the optimal policy should lead to a reward distribution of Pearson shape). In practice, however, the closer the vehicle is to the setpoint the more difficult it becomes for it to physically reduce the errors (which is more akin to a Gaussian distribution). Depending on the system, the operating conditions, and the reward function (among others), the reward distribution can switch between various shapes, potentially making the predefined metric not robust to different distribution assumptions. Thus, we propose to consider a transition as an outlier of interest and to store it in B_2 if its associated reward $r(s_t)$ is:

$$r(s_t) > \mathbb{E}[r(s_t)], \tag{27}$$

where the expected value $\mathbb{E}[r(s_t)]$ is computed over the last 50,000 rewards generated that are stored as an additional variable M. The size of M was chosen in order to compute the expected reward over a moving window of approximately 100 episodes in order to give more importance to novel inputs, similarly to biological systems [13].

This choice of expected value as a metric is related to the subtracted baseline in Eq. (26) that is the Advantage function $A(s, a)$. This function represents the benefits of changing the current policy. If the value of $A(s, a)$ is positive, then the probability to take the evaluated action (with respect to. the given state) will be increased because its Q-Value is higher than the expected one (with respect to. the current policy). Our assumption is that transitions that meet the criteria expressed in Eq. (27) are associated with positive values of $A(s)$. Gradient updates using samples from the Optimistic Memory will therefore mostly improve the expected return of the policy, leading to faster discovery of successful trajectories.

The maximum size of B_2 is set to $10,000$. It is drastically smaller than $B1$ because, as the agent's performance improves over the course of training, what was considered as a good transition is most likely to be outdated. Therefore, the reduced buffer size ensures that we focus on the current best transitions. Finally, contrary to the first buffer, we propose to sample n uncorrelated items from B_2 as single transitions are iteratively stored in this buffer.

V. Training

This section describes the training settings (including training scenario, reward function, state vector, and exploration strategy) with the hyperparameters choice. It ends with an analysis of the training results.

A. Scenarios definition

The training consists of performing a total of 3000 episodes (as no further notable improvement in setpoint regulation is observed after approximately 2500 episodes). The maximum length of a training episode was set at 500 timesteps (equivalent to 25 seconds). We define a training episode as follows:

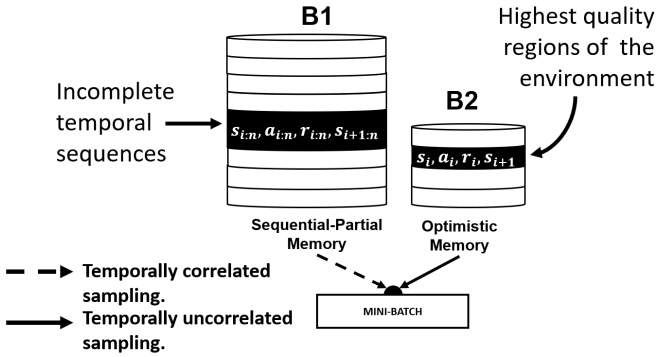


Fig. 1. Illustration of our bio-inspired procedure. The agent's past experience is divided in two memory units: 1 out of 2 transitions are stored in the first buffer and we sample from it temporal sequences of experiences; we store in the second buffer the best transitions as measured by reward and with respect to the current policy. This procedure takes advantage of the resilience from on-policy sampling while keeping the data efficiency from the off-policy formulation.

1) The robot is initialized at a depth of 40 meters with a random orientation $(\psi, \theta, \phi) \in [\frac{-\pi}{4}; \frac{\pi}{4}]$ and at null velocity.

2) The current variables are randomly chosen such that $v_c \in]0, 0.5]$ and $[h_c, j_c] \in [\frac{-\pi}{4}; \frac{\pi}{4}]$ (which remain fixed during the episode) and a random vector of setpoints is generated such that $x_{ref} = [v_x, 0, 0, 0, 0, 0]^T$ with $v_x \in [0.1, 0.5]$ $(m.s^{-1})$. 3) Then, the off-policy $\pi_\mu(a|s)$ behavior is used.

4) The episode ends when the control objective (2) is met or when the episodic step number exceed 500.

B. Reward shaping

Compared to our previous work [30], the control objective treated here is fundamentally more complex since we are not trying to reach one singular state but rather a sequence of desired successive states. We propose the following terminal reward signal to take this into account:

$$r_{success} = 1000 \quad \textbf{if} \quad \forall\ t \in [t{-}100, t],\ |e_i(t)| \leq \chi. \tag{28}$$

Therefore, performing the desired control objective, as shown in Eq. (2), will generate this reward, Eq. (28), and its value was chosen in order to make sure that, for all trajectory lengths, the maximum sum of return is obtained only by stabilizing the vehicle. Otherwise, the reward $r(s_t)$ is generated. Let's define the Euclidean distance to the desired setpoint as $e_{L2}(t) = \sqrt{\sum_{i=1}^{i=dim(u)} e_i^2(t)}$ and its derivative is computed over the last two frames and denoted as $d_{rate}(t)$. The reward $r(s_t)$ is then defined as:

$$r(s_t) = C_1 \times \exp\big[- e_{L2}^2(t) \times C_2 \big] \tag{29}$$

The performance of SAC is highly dependent on the choice of reward scale (or amplitude) which, in the case of our reward function Eq. (29), is controlled by the constant C_1. The reward scale can be interpreted as the inverse of the temperature parameter α from (19) which controls the stochasticity of the resulting policy. Here, we empirically chose $C_1 = 40$, which gave us the best performance, by following advice from [20]. The reward signal, Eq. (29), is equal to its maximum value possible per step (that is C_1) only when all the current errors are equal to zero. As AUVs move slowly, successive states display error signals $e_i(t)$ of minor and similar amplitude. We find that this addition of $C_2 = 10$ (compared to our previous work [15]), makes it easier for the critics to differentiate the State-Value of successive states without altering the reward scale as $\lim_{x \to 0} C \times e^x = C$. This reward function, Eq. (29), encourages the agent to reduce the errors as much and as fast as possible and the vehicle stabilization is further promoted by generating the maximum reward possible per step, as shown in Eq. (28). In order to improve the exploration ability provided by the maximum entropy framework, we used adaptive parameter noise [31] which consists in adding noise to the parameters of the policy network.

C. Process Observability

At each timestep, the agent captures an observation vector o_t representing the process dynamics that we defined as:

$$o_t = [a_{t-1}\,;\, \Theta;\, V\,;\, \Omega;\, u_t\,;\, e_t\,;\, e_{L2}\,;\, d_{rate}\,;\, \delta_\chi], \tag{30}$$

where $a_{t-1} \in \mathbb{R}^{18}$ are the last action estimated (i.e. poles value); $\Theta = [\phi; \theta; \psi]$ are the Euler orientation of the vehicle (roll, pitch and yaw respectively); $V = [v_x; v_y; v_z]$ and $\Omega = [\omega_\phi; \omega_\theta; \omega_\psi]$ are its linear and angular velocities; $u_t \in \mathbb{R}^6$ are the last control inputs applied; $e_t \in \mathbb{R}^6$ are the error values on each setpoint; e_{L2} and d_{rate} as described in Section V-B; and $\delta_\chi \in [0, 1]$ is a variable which keeps track of the number of successive steps where all the errors are within the threshold (i.e. if $\delta_\chi = 1$, the control objective is achieved). The dimension of the observation vector o_t is therefore equal to 42. Noted that with this observation vector (30), the current disturbance characteristics are not included. In order to improve the process observability and following our previous results [32], we construct our state vector s_t out of the current and past observation vectors along with their two-by-two difference. This results in a 126 dimensional state space defined as $s_t = [o_t\ ;\ o_{t-1}\ ;\ o_{t-1} - o_t]$.

D. Training results

Figure 2 shows the training dynamics with the normalized mean return per episode, the RMSE on the setpoint per episode, and the normalized mean return standard deviation per episode. The yellow dashed lines represent the performance of the OFP controller. It is fair to compare these controllers, as they are based on the exact same control structure as our learning-based controller, that is the PID structure. Thus, the only difference between the three controllers is the value of the poles used to compute the PID control inputs.

As we can see in Figure 2, both methods are able to learn the task and converge toward what seems to be a maximum value of the reward. In the third plot of Figure 2, we can see that the performance improvement is smoother with the BIER method which exhibits a lower reward standard deviation (which tends to reduce over time contrary to the CER agent). With the CER method, the variance is higher, with pikes that even drive the agent to lower performance than those obtained with the OFP controller. Noted that the OFP controller displays the lowest

standard deviation (third plot of Figure 2), thanks to the model information incorporated in SMAC [17].

The agent trained using the BIER method was able to exceed the performance of the OFP controller, in less than half the number of episodes compared to the standard CER method. It is represented in Figure 2 by the vertical lines. Therefore, when DRL can be considered on the physical platform, a designer might favor the use of our BIER method for improved data efficiency and learning stability.

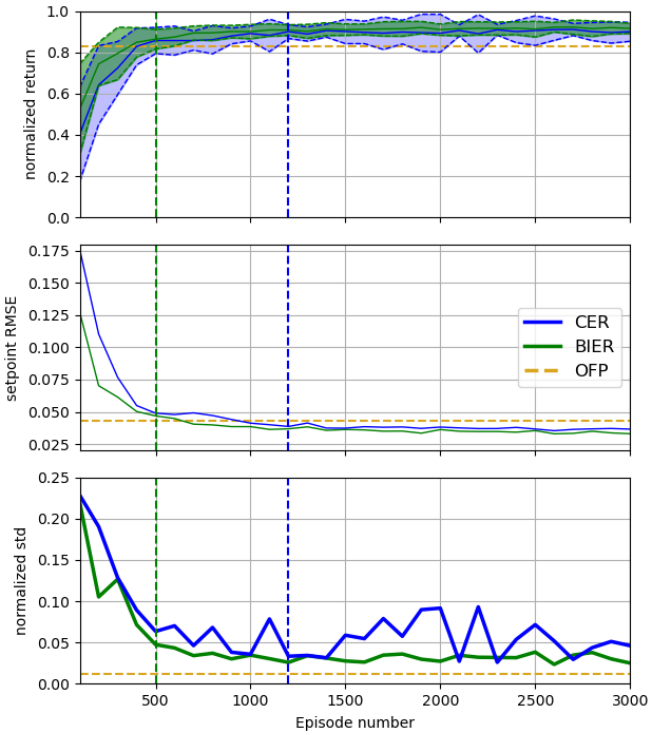


Fig. 2. Training curves for both Experience Replay methods. The BIER agent outperforms the CER agent in both learning speed and variance. Both learning-based controllers outperform the model-based controller (i.e. OFP).

VI. Experiments and Analysis

The challenge of conducting DRL in real-life lies in the drop in performance caused by the distribution shift that is here the difference between the training and evaluation set of states. In order to visualize this behavior in our use case, we present below different evaluation scenarios of increasing complexity as measured by desired setpoint and current velocity.

Scenario 1: the setpoint range is the same as during training but no current disturbances is applied. This scenario is therefore in theory simpler than the training one.

Scenario 2: the process is the same as during training but with setpoint and current variables that were not seen during training (but are still in the same range).

Scenario 3: the setpoint v_x is increased to the range [0.5,1.0], the current variables are the same as during training.

Scenario 4: the setpoint characteristic is the same as during training but the current velocity v_c is increased to [0.5,1.0] $(m.s^{-1})$ and $(h_c, j_c) \in [-\pi, \frac{-\pi}{2}[\cup]\frac{\pi}{2}, \pi]$.

Scenario 5: both the setpoint and current variables are increased to the range defined in scenarios 3 and 4.

Scenario 6: we keep the increased range from scenario 5, and at a random timestep during the episodes (between the 100th and 400th timestep), we vary the current variables (velocity and orientation) within the same training range.

In Table I, the metrics were computed over 500 episodes (different from each other) for each evaluation scenario. The line "Baseline" denotes the performance obtained at the end of the training, which incorporates the parameter noise described in Section V-B. For the other scenarios, this noise is removed. In order to visualize more easily the performance variation, we propose in Figure 3 an illustration of these results.

The CER and BIER agents are able to stabilize the vehicle over the first 3 scenarios where the performance is matching the training one. When increasing the desired setpoint value, we can see the associated RMSE slightly increasing but the performance remains satisfying. The performance drop is the largest when increasing the sea current disturbance (i.e. scenarios 4, 5, and 6). The sensibility to this disturbance is further depicted by the performance difference that is much smaller between scenarios 5 and 6 compare to between scenarios 3 and 4. We believe this is due to the current characteristics not being explicitly included in the state vector.

The BIER agent performs better on scenario 1 compared to scenario 2 which is simpler than the training scenario as it does not incorporate sea current disturbance. More interestingly, its performance compared to CER has also increased on scenarios 5 and 6, despite the distribution shift being particularly large there. This suggests that the policy obtained with the BIER method has better generalization abilities. We can see that the OFP controller is much sensitive to sea current disturbance compare to setpoint variation (as it is included in the model). Noted that the learning-based controllers are able to exceed the OFP controller on scenario 6, despite not having experience such disturbance during training.

These results show that ER does have an impact on the quality of the resulting policy. By only manipulating differently the agent's past experience, we were able to make learning faster, and with improved robustness to increased process variations. Reducing the distribution shift problem in DRL remains an active field of research and future work could focus i) on improving process observability and ii) studying how conservative can a DRL-based policy be.

VII. Conclusion

In this article, we proposed a new Bio-Inspired Experience Replay which aims at incorporating concepts from the biological *Replay Mechanism* in the context of DRL. We found that by taking inspiration from nature, while keeping in mind the requirements from Gradient-based optimization, the adaptive controller can learn faster and with improved stability. Our results also suggest that learning-based adaptive control might be a key ingredient toward real-life autonomous robotic applications, by taking advantage of the available information on the process, with a model-based structure, and using model-free learning to adjust its parameters to compensate for unmodeled process variation, via Deep Reinforcement Learning.

TABLE I
EVALUATION RESULTS FOR THE CER METHOD (LEFT) AND THE PROPOSED BIER METHOD (RIGHT).

Scenario	mean RMSE per step	normalized mean return	mean RMSE per step	normalized mean return
Baseline	0.0364	0.9104 ± 0.0461	**0.0330**	**0.9219 ± 0.0250**
1	0.0370	0.9072 ± 0.0309	**0.0366**	**0.9347 ± 0.0262**
2	0.0350	0.9108 ± 0.0456	**0.0320**	**0.9244 ± 0.0240**
3	0.0448	0.8774 ± 0.0416	**0.0418**	**0.9124 ± 0.0266**
4	0.1483	0.4078 ± 0.2965	**0.1214**	**0.5071 ± 0.2530**
5	0.1656	0.3556 ± 0.2846	**0.1508**	**0.4289 ± 0.2573**
6	0.1802	0.3238 ± 0.2219	**0.1637**	**0.3966 ± 0.2167**

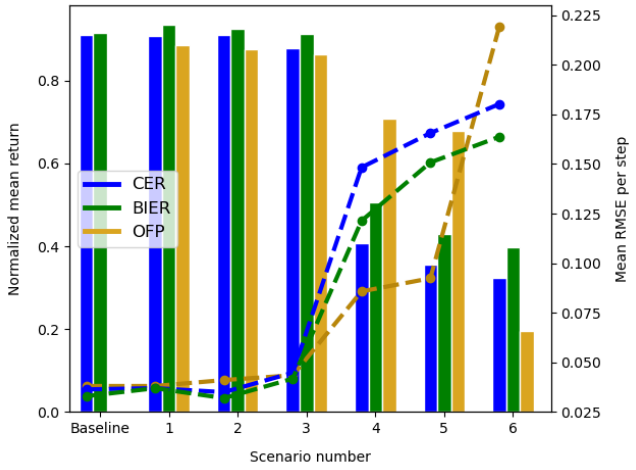


Fig. 3. Illustration of the evaluation performance. The BIER agent performs consistently better than the CER one despite having been trained using a reduced variety of samples.

REFERENCES

[1] K. Astrom and B. Wittenmark, *Adaptive Control*. Dover Publications, 2008.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning an introduction - Second edition*. MIT Press, 2018.

[3] S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments," *ACM Comput. Surv.*, vol. 54, no. 6, 2021.

[4] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[5] R. Yu, Z. Shi, C. Huang, T. Li, and Q. Ma, "Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle," *2017 36th Chinese Control Conference (CCC)*, 2017.

[6] C. Wang, L. Wei, Z. Wang, M. je Song, and N. Mahmoudian, "Reinforcement learning-based adaptive trajectory planning for auvs in under-ice environments," *OCEANS 2018 MTS/IEEE Charleston*, 2018.

[7] K. B. Knudsen, M. C. Nielsen, and I. Schjølberg, "Deep learning for station keeping of auvs," *OCEANS 2019 MTS/IEEE Seattle*, 2019.

[8] Z. Chu, B. Sun, D. Zhu, M. Zhang, and C. Luo, "Motion control of unmanned underwater vehicles via deep imitation reinforcement learning algorithm," *Iet Intelligent Transport Systems*, vol. 14, pp. 764–774, 2020.

[9] L. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, 2004.

[10] S. Zhang and R. Sutton, "A deeper look at experience replay," *arXiv preprint arXiv:1712.01275*, 2017.

[11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *CoRR*, vol. abs/1511.05952, 2016.

[12] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in *International Conference on Machine Learning*, 2020.

[13] T. L. Hayes, G. P. Krishnan, M. Bazhenov, H. T. Siegelmann, T. J. Sejnowski, and C. Kanan, "Replay in deep learning: Current approaches and missing biological elements," *Neural Computation*, vol. 33, no. 11, pp. 2908–2950, 2021.

[14] M. Manhães, S. Scherer, M. Voss, L. Douat, and T. Rauschenbach, "UUV simulator: A gazebo-based package for underwater intervention and multi-robot simulation," in *MTS/IEEE OCEANS*, Monterey, 2016.

[15] T. Chaffre, G. Le Chenadec, K. Sammut, E. Chauveau, and B. Clement, "Direct adaptive pole-placement controller using deep reinforcement learning: Application to auv control," *IFAC-PapersOnLine*, vol. 54, no. 16, pp. 333–340, 2021, 13th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS.

[16] A. O'Dwyer, *Handbook of PI and PID controller tuning rules*. Imperial College Press, 2006.

[17] F. Hutter, H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *LION*, 2011.

[18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[19] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *CoRR*, vol. abs/1812.05905, 2018.

[20] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *arXiv preprint arXiv:1812.11103*, 2019.

[21] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proceedings of the 34th International Conference on Machine Learning, ICML*. JMLR.org, 2017, pp. 1352–1361.

[22] Z. Ahmed, N. L. Roux, M. Norouzi, and D. Schuurmans, "Understanding the impact of entropy on policy optimization," in *ICML*, 2019.

[23] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, 2016.

[24] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

[25] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 2005.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.

[27] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *EEE Hot Chips 20 Symposium (HCS)*, 2008.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR*, Y. Bengio and Y. LeCun, Eds., 2015.

[29] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[30] T. Chaffre, J. Moras, A. Chan-Hon-Tong, J. Marzat, K. Sammut, G. Le Chenadec, and B. Clement, "Learning-based vs model-free adaptive control of a mav under wind gust," in *Informatics in Control, Automation and Robotics*. Springer, 2022, pp. 362–385.

[31] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," in *6th International Conference on Learning Representations, ICLR*, 2018.

[32] T. Chaffre, J. Moras, A. Chan-Hon-Tong, and J. Marzat, "Sim-to-real transfer with incremental environment complexity for reinforcement learning of depth-based robot navigation," in *17th International Conference on Informatics, Automation and Robotics, ICINCO*, 2020.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

Title: Reinforcement Learning and Sim-to-Real Transfer for Adaptive Control of AUV

Keywords: Adaptive Control, Deep Reinforcement Learning, Robotics, Underwater Vehicles.

Abstract: Autopilots for unmanned systems are usually designed based on the feedback provided by velocity and orientation sensors. In the case of autopilot systems for autonomous underwater vehicles (AUVs), the main objective in the design is to compensate for waves and current-induced disturbing forces acting on their body. Existing AUV autopilots are however only able to compensate for low-frequency components of sea-induced disturbances. It seems natural to assume that the AUV performance could be improved by taking the nature of the disturbances into account in the design of the autopilot.
Adaptive control provides what seems to be an ideal framework for this end. The objective of this technique is to adjust automatically the control parameters when facing unknown or time-varying processes such that the desired performance threshold is met. Developed in the late 1950s, adaptive control frameworks have been considerably expanded and used in various fields, their application has been facilitated by the rapid progress in microelectronics and the increasing interaction between laboratories and companies, from aerospace to maritime industries. As a result, adaptive controllers started to be widely adopted in the industry in the early 1980s. It was established at that time that *robust designs* with fixed parameters are too limited to handle complex regimes. The study of adaptive controllers for AUV maneuvering is associated with various challenges, and the focus of this thesis was the external disturbances including:

Unknown dynamics: the uncertainty associated with describing precisely the states of waves or currents is high. This, together with its dynamic nature, prevents linear feedback control methods from achieving optimal performance of the plant. This becomes more critical in the presence of changes in weather conditions that impose a multiplicative factor in the component of the induced forces. The disturbance period will also vary with the speed of the vehicle and its orientation relative to the waves.

Nonlinearity: the controller response at some operating points must be overly conservative to satisfy the specification at other operating points. This is difficult to achieve for fixed parameters obtained through local linearization, that do not encompass the entire regime envelope.

In this thesis, we considered the case where the AUVs have limited observability of the process and therefore the aforementioned uncertainties are not measured by the system. A class of adaptive control methods, known as learning-based adaptive controllers, have been developed to tackle some of these limitations. This family of solutions uses model-free optimization methods capable of compensating for the unknown part of a process while also maintaining optimal control of its known part using traditional model-based control structures. Among the various model-free methods, deep reinforcement learning is currently leading the field. They exploit strong statistical tools that provide control systems the ability to automatically learn and improve from experience without being explicitly told how to.

The objective of this thesis was to formalize a novel learning-based adaptive control using deep reinforcement learning and adaptive pole-placement control. In addition, we proposed a novel experience replay mechanism that takes into account the characteristic of the biological replay mechanism. The methods were validated in simulation and in real life, demonstrating the benefits of combining both theories against using them separately.