From learning-based identification to model-based control of robotic systems

Sardor Israilov

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

De l'identification basée apprentissage profond à la commande basée modèle

Sardor ISRAILOV

Laboratoire d'Informatique, de Signaux et Systèmes de Sophia Antipolis (I3S)
UMR7271 UCA CNRS

Présentée en vue de l'obtention du grade de docteur en AUTOMATIQUE, TRAITEMENT DU SIGNAL ET DES IMAGES
d'Université Côte d'Azur

Dirigée par : Guillaume ALLIBERT, Professeur, Université Côte d'Azur
Co-dirigée par : Médéric ARGENTINA, Professeur, Université Côte d'Azur

Devant le jury, composé de :
Christophe ELOY, Professeur, Aix-Marseille Université
Frédéric BOYER, Professeur, IMT Atlantique
Massimo CENCINI, Professeur, CNR-ISC
David FILLIAT, Professeur, ENSTA Paris - Institut Polytechnique de Paris
Li FU, Maître de conférences, École Centrale de Lyon
Christophe RAUFASTE, Professeur, Université Côte d'Azur
Soutenue le : 30/01/2024

DE L'IDENTIFICATION BASÉE APPRENTISSAGE PROFOND À LA COMMANDE BASÉE MODÈLE

From learning-based identification to model-based control of robotic systems

Sardor ISRAILOV

Jury :

Président du jury
Christophe ELOY, Professeur, Aix-Marseille Université

Rapporteurs
Frédéric BOYER, Professeur, IMT Atlantique
Massimo CENCINI, Professeur, CNR-ISC

Examinateurs
David FILLIAT, Professeur, ENSTA Paris - Institut Polytechnique de Paris
Li FU, Maître de conférences, École Centrale de Lyon

Directeur de thèse
Guillaume ALLIBERT, Professeur, Université Côte d'Azur

Co-directeur de thèse
Médéric ARGENTINA, Professeur, Université Côte d'Azur

Membres invités
Christophe RAUFASTE, Professeur, Université Côte d'Azur

Université Côte d'Azur

Sardor ISRAILOV

De l'identification basée apprentissage profond à la commande basée modèle

xii+160 p.

Résumé

La nage des poissons reste un sujet complexe qui n'est pas encore totalement compris en raison de son aspect interdisciplinaire qui mêle la biologie et dynamique des fluides. Au fil des millénaires, les organismes naturels ont perfectionné leur biologie pour naviguer efficacement dans leur environnement et s'adapter à tout type de situations. Tout au long de l'histoire, l'humanité s'est inspirée de la nature pour innover et développer des systèmes biomimétiques. Le poisson robotique, en particulier, trouve nombres d'applications dans le monde réel et son contrôle doit encore être optimisé. L'apprentissage par renforcement profond a donné d'excellents résultats dans le contrôle des systèmes robotiques, dont la dynamique est trop complexe pour être entièrement modélisée et analysée. Dans cette thèse, nous avons exploré de nouvelles voies de contrôle d'un poisson biomimétique via l'apprentissage par renforcement afin de maximiser efficacement la force de poussée et la vitesse de déplacement. Cependant, pour comprendre pleinement ces nouveaux algorithmes basés sur les données, nous avons d'abord étudié l'application de ces méthodes sur une référence standard de la théorie du contrôle, le pendule inversé sur un chariot. Nous avons démontré que l'apprentissage par renforcement profond pouvait contrôler le système sans aucune connaissance préalable du système, en obtenant des performances comparables aux méthodes traditionnelles de la théorie du contrôle basée sur un modèle. Dans le troisième chapitre, nous nous concentrons sur la nage ondulatoire d'un poisson robotique avec différents objectifs et sources d'information de contrôle. Nos études indiquent que la force de poussée d'un poisson robotique peut être optimisée en utilisant des données provenant à la fois de capteurs de force et d'une caméra comme retour d'information pour la commande. Nos résultats démontrent qu'une commande carrée avec une fréquence particulière maximise la poussée et nous la rationalisons en utilisant le principe du maximum de Pontryagin. Un modèle approprié est établi qui montre un excellent accord entre la simulation et les résultats expérimentaux. Ensuite, nous nous concentrons sur la maximisation de la vitesse d'un poisson robotique à la fois dans plusieurs environnements virtuels et dans des expériences utilisant des données visuelles.

Mots-clés : Nage de poisson | Mécanique des fluides | Théorie du contrôle | Apprentissage automatique | Contrôle optimal | Apprentissage par renforcement | Réseaux neuronaux | Pendule inversé | Poisson robotisé | Optimisation

Abstract

Fish swimming remains a complex subject that is not yet fully understood due to the intersection of biology and fluid dynamics. Through years of evolution, organisms in nature have perfected their biological mechanisms to navigate efficiently in their environment and adapt to particular situations. Throughout history, mankind has been inspired by nature to innovate and develop nature-like systems. Biomimetic robotic fish, in particular, has a number of applications in the real world and its control is yet to be optimized. Deep Reinforcement Learning showed excellent results in control of robotic systems, where dynamics is too complex to be fully modeled and analyzed. In this thesis, we explored new venues of control of a biomimetic fish via reinforcement learning to effectively maximize the thrust and speed. However, to fully comprehend the newly-emerged data-based algorithms, we first studied the application of these methods on a standard benchmark of a control theory, the inverted pendulum with a cart. We demonstrated that deep Reinforcement Learning could control the system without any prior knowledge of the system, achieving performance comparable to traditional model-based control theory methods. In the third chapter, we focus on the undulatory swimming of a robotic fish, exploring various objectives and information sources for control. Our studies indicate that the thrust force of a robotic fish can be optimized using inputs from both force sensors and cameras as feedback for control. Our findings demonstrate that a square wave control with a particular frequency maximizes the thrust and we rationalize it using Pontryagin Maximum Principle. An appropriate model is established that shows an excellent agreement between simulation and experimental results. Subsequently, we concentrate on the speed maximization of a robotic fish both in several virtual environments and experiments using visual data. Once again, we find that deep Reinforcement Learning can find an excellent swimming gait with a square wave control that maximizes the swimming speed.

Keywords: Swimming | Fluid mechanics | Control theory | Machine Learning | Optimal Control | Reinforcement Learning | Neural Networks | Inverted Pendulum | Cart-pole | Robotic Fish | Optimization

Acknowledgements

Undertaking this PhD has been a profound and life-forming experience that broadened my knowledge and view of life. I am sincerely grateful to have gone through this journey and to have experienced these three years of formation that profoundly impacted me in the way I think, write and communicate. I would like to thank Digital Systems for Humans (DS4H) of Université Côte d'Azur for financing my thesis and providing me with such an amazing opportunity. This journey would not be possible without many people who provided me with guidance and support.

First of all, I would like to thank Médéric Argentina and Guillaume Allibert for providing me guidance and support during my thesis. I would also thank other members of the project : Li Fu, Christophe Raufaste and Franco Fusco for providing valuable advice on tackling different challenges arising during my thesis. Thank you all for the patience and advice you have provided me to conduct my thesis.

I would like to thank Massimo Cencini and Frédéric Boyer for accepting the report for my defense. I would like to thank Christophe Eloy and David Filliat for accepting to be examiners at my thesis.

As I am part of two research labs, I have got to know many amazing people. I would like to thank my colleagues for all the time we spent during lunch and coffee breaks : Giulia Rocco, Quentin Guimard, Arnab Dey, Dalia Hareb, Anderson Lourenco de Araujo, Houssem Boulahbal, Tarek , Tomais Jonas Konrad, Ninad Manerikar and Yacine Khacef from I3S and David Paulovics, Martial Morisse, Luis Alberto Razo López, Hector Letellier, Juliette Huynh, Apoorva APOORVA, Gianni Aupeetit Diallo and Pierre Azam from Inphyni. I would also like to thank my friends that brought me joy during my phd : Piotr Krasnowski, Franco Fusco, Sofia Ahmed Ahamada Yann fraboni and Julie Salles.

I have had a truly amazing time in Nice thanks to amazing hikes and people that I did it with, particularly "Mountain gang group" : Damien, Abder, Antoine, Geremia, Michal, Mehdi, Thomas, Ivana and many others. Thanks to you all, I had an amazing time in the mountains and fell in love with the region.

I would like to thank my friends and family who have been there for me during my hard times and Covid period. I would like to thank Abdul-Aziz Israilov and Antonina Studenikina for the moral support and inspiration that they have given me. I would like to thank my parents and grandparents for the hope and belief I needed to persevere my thesis both in joyful and hard moments.

Table of contents

Annexes

Notations

Symbol | Phrase	Meaning
ODE	Ordinary differential equation
linear system	system in which the change of the output is linearly proportional to the change of the input
nonlinear system	system in which the change of the output is not proportional to the change of the input
Linearization	method of approximating a nonlinear system with a linear system
LTIS	Linear Time Invariant System
setpoint	reference position for control
$s \in \mathcal{S}$	States.
$a \in \mathcal{A}$	Actions.
$r \in \mathcal{R}$	Rewards.
s_t, a_t, r_t	State, action, and reward at time step t of one trajectory.
γ	Discount factor; penalty to uncertainty of future rewards; $0 < \gamma \leq 1$.
R_t	Return; or discounted future reward; $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
$P(s', r \mid s, a)$	Transition probability of getting to the next state s' from the current state s with action a and reward r.
$\pi(a \mid s)$	Stochastic policy (agent behavior strategy); $\pi_\theta(.)$ is a policy parameterized by θ.
$\mu(s)$	Deterministic policy; we can also label this as $\pi(s)$, but using a different letter gives better distinction.
$V(s)$	State-value function measures the expected return of state s; $V_w(.)$ is a value function parameterized by w.
$V^\pi(s)$	The value of state s when we follow a policy π; $V^\pi(s) = \mathbb{E}_{a\sim\pi}[G_t \mid S_t = s]$.
$Q(s, a)$	Action-value function or quality value is similar to $V(s)$, but it assesses the expected return of a pair of state and action (s, a); $Q_w(.)$ is an action value function parameterized by w.
$Q^\pi(s, a)$	Similar to $V^\pi(.)$, the quality value of (state, action) pair when we follow a policy π; $Q^\pi(s, a) = \mathbb{E}_{a\sim\pi}[G_t \mid S_t = s, A_t = a]$.
$A(s, a)$	Advantage function, $A(s, a) = Q(s, a) - V(s)$; it can be considered as another version of Q-value with lower variance.

CHAPTER 1

Introduction and context

Mankind has always been fascinated by the vast and mysterious sea realm that has yet to be fully explored. Around 71% of the earth's surface is water-covered, yet more than 80% of the ocean surface is still not mapped [Eakins and Sharman, 2010]. Oceans are full of undiscovered natural resources and humanity has made significant strides in uncovering these resources [Lubchenco and Haugan, 2023]. A lot of natural resources still remain in unexplored places of the sea. It is inevitable that we develop technologies to navigate and fully explore the oceans. This puts humanity on the path of a growing industrial interest in submarines, UAVs (Underwater Autonomous Vehicles), shipbuilding and cruise industry.

The technological and industrial progress in recent years has made it possible for us to interact with the marine environment in different forms of aquatic endeavors. There has also been a surge in demand for the development of submarines and UAVs, primarily for military defense applications. This shift has substantially influenced the economic landscapes of numerous nations and industries.

Nature is a living system that has been managing its resources efficiently and continuously for millions of years. Through years of evolution, animals, plants, and other living beings have developed biological mechanisms to cope with the challenges of their environment and ensure their survival [Gayon, 1998]. This shows that nature is a source of inspiration for sustainable design and makes nature an excellent model for imitation in many aspects of our lives [Sanchez et al., 2005]. Nature has been a rich source of inspiration for numerous technological innovations, encompassing materials, sensors, robotics, and more [Farhat et al., 2019, Triantafyllou and Triantafyllou, 1995, Koo et al., 2018]. Among the variety of organisms that inspire robotic design, fish are particularly promising due to their ability to navigate a vast range of aquatic environments with precision and efficiency. Their swimming capabilities, streamlined bodies, and sensory feedback systems are testaments to millions of years of evolutionary development.

To enhance our exploration of the world's oceans, it is imperative to study and understand the behavior of their primary residents, fish. Fish have energy-efficient and maneuverable swimming patterns. Their efficiency is so great that it surpasses the capabilities of human-designed underwater vehicles.

A minimally disruptive way of observing marine life is especially useful when studying animals' behaviors, swimming gaits and interactions within their natural environment [Krause et al., 2011]. One way to achieve this is to use UAV that could swim alongside marine life to allow close-range examinations. Underwater vehicles predominantly use propellers or jet-based propulsion systems. This kind of propulsion system can potentially scare marine life and disturb up-close

observations [Bruzzone et al., 2021]. Many of these traditional UAV designs are not only cumbersome, but also demand complex and expensive fabrication processes. The robotic fish is a natural candidate to observe marine life by mimicking the fish movement without disturbing them. Due to its hydrodynamic shape and fish resemblance, it can easily fit into the aquatic animal kingdom. The observations of marine life are crucial to study the underwater ecological system and the impact of climate change on it, providing insights to mitigate this effect. It has also been shown that fish can also follow fish like robots that can evacuate them from dangerous zones impacted by pollution such as oil spill in the ocean [Polverino et al., 2013]. Not only can a biomimetic robotic fish observe the marine life of real fish without disturbing them, but also a soft robotic fish can be a leader in a school of natural fish [Marras et al., 2012] and bring them to different locations.

One of the primary motivations of constructing a robotic fish is that fish has superior efficiency and agility with respect to UAVs . It is shown that the propulsion efficiency of the fish can exceed 90% [Triantafyllou and Triantafyllou, 1995]. The robotic fish is also small in its cross section, which permits it to access the hardly accessible zones underwater. There have been numerous applications of robotic fish for observing pollution or wild nature underwater [Kohnen, 2009]. One of those [Hu et al., 2011] tracks water pollution with the swarm of robotic fish equipped with sensing capabilities.

1.1 Aquatic Locomotion

The movement of a swimmer through water results from interaction of animal dynamics with the surrounding fluid. This fluid-structure interaction (FSI) case is hard to resolve in a closed form for several reasons. Firstly, the deformation of a swimmer is hard to model mechanically and the loads and stresses are barely calculable given a soft material of interaction. Soft tissue deformations can be approximated and the general physics frameworks applied to tackle the challenge, such as *large deformation theory* [Dill, 2006] or the classical *beam theory* [Cheng et al., 1998, Pedley and Hill, 1999, Ramananarivo et al., 2013].

Aquatic animals generate velocity and pressure fields while moving in water. The knowledge of pressures and velocities permits us to analyze the Navier-Stokes equations [Landau and Lifshitz, 1987] which are partial differential equations (PDE) that can describe the motion of Newtonian and in-compressible fluids and gases in three dimensions. The first equation is the mass balance equation :

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0, \tag{1.1}$$

where ρ is the density of the fluid and $\boldsymbol{u}$ is a velocity vector in three dimensions. This equation links density change to the mass flux. Because fluid is incompressible and matter is conserved, from the differential form of continuity equation reduces to :

$$\nabla \cdot \boldsymbol{u} = 0. \tag{1.2}$$

Also, according to Newton's second law ($\sum \boldsymbol{F} = m\boldsymbol{a}$), when fluid is in motion, the momentum is conserved :

$$\underbrace{\rho \left(\frac{\partial}{\partial t} + \boldsymbol{u} \cdot \nabla \right) \boldsymbol{u}}_{internal forces} = \underbrace{-\nabla p}_{pressure} + \underbrace{\mu \nabla^2 \boldsymbol{u}}_{viscous} + \underbrace{g}_{external}, \tag{1.3}$$

where μ measures the fluid viscosity and g represents external forces such as gravity. Furthermore we neglect the external forces ($g = 0$), because fish are almost neutrally buoyant. These are the

fundamental and general PDEs that are used in the simulation of many fluid systems from analyzing the flight of airplanes to understanding ocean currents that could result in natural disasters. The Eq. (1.3) also incorporates the pressure and viscous terms. Depending on the situation, one term is dominant over another. A dimensionless number known as the Reynolds Number (Re) is introduced to nondimensionalize Eq. (1.3). Defining characteristic parameters, the length of the animal with L, and the locomotion speed of a swimmer with U , the Reynolds Number is defined as the ratio of inertial terms to the viscous forces :

$$Re \sim \frac{\rho(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}}{\mu\nabla^2\boldsymbol{u}} \sim \frac{\rho U^2/L}{\mu U/L^2} = \frac{\rho UL}{\mu}. \tag{1.4}$$

The Reynolds Number permits to determine whether the flow induced by the swimmer is turbulent, laminar or in-between. If $Re \ll 1$, then the viscous forces are dominant over the inertial terms and the flow is said to be stokesian. In this limit, inertial terms are negligible with respect to the viscous terms, then Eq. (1.3) can be reduced to Stokes equations which can be solved analytically in steady state :

$$\nabla p = \mu\nabla^2\boldsymbol{u} \tag{1.5}$$

Note that Eq. (1.5) does not depend on time. The region, where $Re \sim 1$ correspond to the region of small microscopic creatures living in the fluid, from several millimeters to several centimeters (for example, larvae).

In the general case, when $Re \gg 1$ in (Eq. (1.3)), Navier-Stokes equations are hard to solve analytically and turbulence appears for large Reynolds numbers (for instance, large cruise ships swimming in the ocean). The general solution of Eq. (1.3) and its uniqueness is still an unsolved problem and is part of one of the seven millennium prize problems [Devlin, 2002].

The eight orders of magnitude in Reynolds number of aquatic locomotion is shown in Fig. 1.1. These animals vary in length from a few centimeters to 30 meters that can reach blue whales.

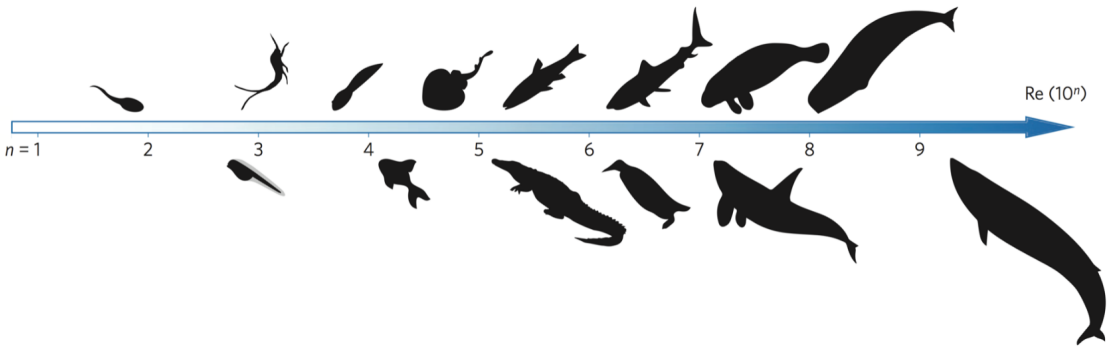

Figure 1.1 – aquatic swimmers and the corresponding Reynolds number, gathering larvae, fish, amphibians, reptiles, marine birds and large mammals. From [Gazzola et al., 2014]

Among the aquatic swimmers, there is a big abundance of ways of locomotion. Some animals like jellyfish eject the water from their internal body chambers to move forward, while other animals like aquatic birds or certain amphibians move their limbs like paddles to move forward [see Fig. 1.2]. Their motion models can be found in [Alexander, 2013]. The four general types of aquatic locomotion can be found in Fig. 1.2.

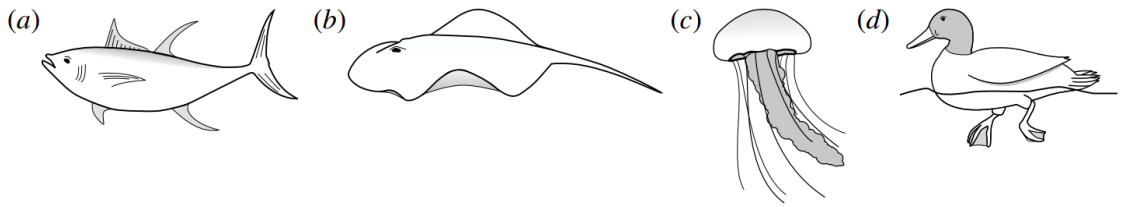

Figure 1.2 – Types of aquatic locomotion (a) oscillatory : tuna, (b) undulatory : ray, (c) pulsatile jet : jellyfish and (d) drag based : duck. From [Van Buren et al., 2019]

Among the aquatic animals, there is a big group constituted mainly from fish. These fish primarily employ one of two principal locomotion mechanisms [see Fig. 1.2a and b]. The first group [Fig. 1.2a] only involves the movement of the caudal fin, termed oscillatory motion [Vatanabe et al., 2008] (oscillatory). The second mechanism [Fig. 1.2b] involves the movement of their spines propagating deformation waves through the body to propulse [Lauder and Tytell, 2005] (undulatory). In nature, most of fish (around 80%) exhibit undulatory motion [Di Santo et al., 2021].

1.2 Fish Locomotion

Understanding the fundamentals of fish locomotion is a key to designing efficient artificial underwater systems. Richard Bainbridge revolutionized the understanding of swimming mechanisms [Bainbridge, 1958] in the 1950s. His pioneering work was one of the first to comprehend the relationship between the fish kinematic characteristics (tip-to-tip tail beat amplitude A and frequency f) and the swimming speed, U. His work gave the general relation linking fish' length L, frequency f and velocity U :

$$\frac{U}{L} = \frac{3}{4}f - 1$$

He also found that in general, the ratio between the tail amplitude and the length of a fish remained constant :

$$\frac{A}{L} \approx 0.2 \tag{1.6}$$

We now construct the relationship between kinematic characteristics and the swimming speed from simple physical derivations [see Fig. 1.3]. When fish undulates its tail, it generates the thrust

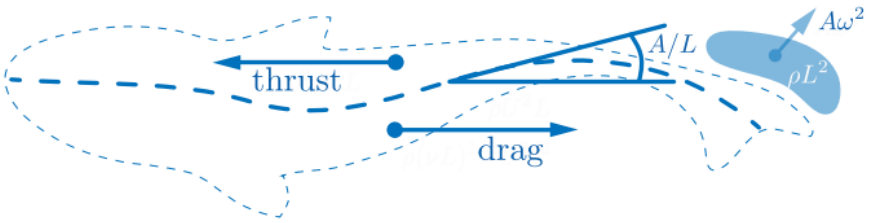

Figure 1.3 – Schematic for fish locomotion. Basic kinematic variables are represented. From [Gazzola et al., 2014]

force and is subject to drag that opposes the movement. The caudal fin of a fish propels a volume

of water, with its mass being proportional to the density of the fluid and a power of the fish's length $m \propto \rho L^3$. The water is moving with the acceleration $A.\omega^2$. According to Newton's laws, the fish experiences the force in the opposite direction. The thrust force is equivalent to the projection of this force on the axis of fish movement (in small angle limit : $\sin(A/L) \sim A/L$) :

$$F_{thrust} \sim \rho A^2 \omega^2 L^2. \tag{1.7}$$

While moving, fish are subject to the skin and pressure drag [see Fig. 1.3 and Eq. (1.3)]. Skin drag is the force opposing the movement induced by the viscous friction between the fish skin and the water. The pressure drag is the drag force due to the difference of pressure between the front and the back of the fish in turbulent flow. The dominating drag force depends on the Reynolds number (Re) of fish-water interaction. As it has been shown in [Gazzola et al., 2014], fish related to high Reynolds number ($Re > 10^3 \sim 10^4$) are mainly subject to pressure drag.

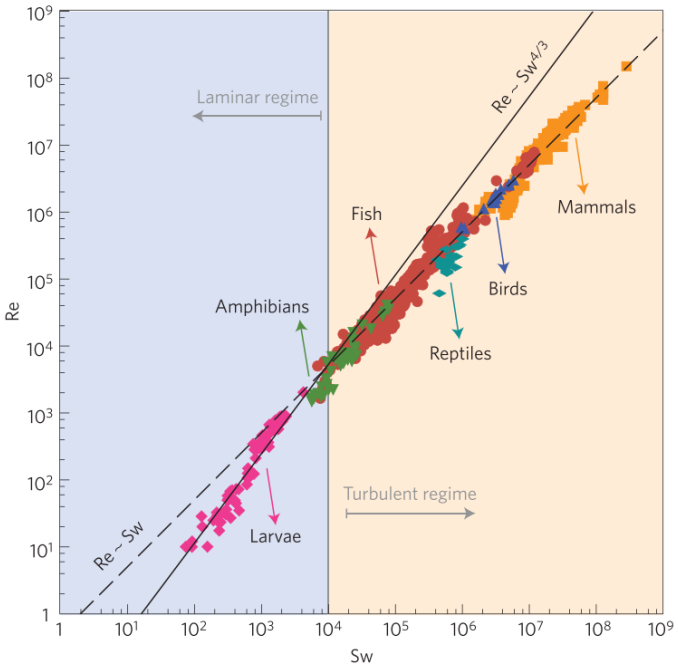

Figure 1.4 – The Reynolds number, as a function of the swimming number ($Sw = \omega AL/\nu$) for various aquatic species, reveals the presence of two distinct regimes for inertial swimmers : the laminar regime and the turbulent regime. From [Gazzola et al., 2014]

$$F_{pressure.drag} \sim \rho U^2 L^2. \tag{1.8}$$

At the cruising speed, the thrust force is equilibrated by the pressure drag which implies that :

$$U \propto A\omega, \tag{1.9}$$

for the fish locomotion, when $Re = \frac{UL}{\nu} > 3000$ [Gazzola et al., 2014] (kinematic viscosity ν). Below Reynolds number $Re_c \approx 3000$, the boundary layer of fish-water interaction becomes dominant; the viscous drag can no longer be neglected and the force balance derivation yields that :

$$U \sim A^{4/3} \omega^{4/3} L^{1/3} \nu^{-1/3}. \tag{1.10}$$

The dimensionless number called Swimming Number, Sw, has been introduced :

$$Sw = \frac{\omega A L}{\nu}, \tag{1.11}$$

to nondimensionalize proportionalities 1.9 and 1.10. The resulting tendencies are :

$$\begin{array}{ll} Re \sim Sw^{4/3}, & Re < 3000 \\ Re \sim Sw, & Re > 3000 \end{array} \tag{1.12}$$

This is the power law of swimming that has been validated both in numerical simulations [Gazzola et al., 2014] and on the experimental data in Fig. 1.4. The way a fish chooses the amplitude A of its tail oscillation has been studied extensively in the literature [Bainbridge, 1958, Sánchez-Rodríguez et al., 2023]. Yet no study demonstrates rigorously how and why fish chooses the swimming gait or frequency ω of its tail undulation in experiments.

Early research in the investigation of swimming gaits was only experimental [Bainbridge, 1958] and the quantity and reproducibility of data was limited. The experimental approach was soon complemented with theoretical models of fish locomotion based on the research from aerodynamic theory. Slender body theory [Munk, 1924] became the basis of Lighthill's elongated body theory [Lighthill, 1971] and thin airfoil theory served as the starting point of Argentina's [Argentina and Mahadevan, 2005, Gazzola et al., 2015] swimming propulsion theories. Even if these analytical models lay a profound foundation in the understanding of fish locomotion, they are limited with an experimental validation. Biomimetic robotic swimmers, on the other hand, serve as an experimental setup for testing the hypotheses and models of swimming propulsion [Ramananarivo, 2014, Gibouin et al., 2018, Sánchez-Rodríguez et al., 2021]. The Robo-Tuna [Triantafyllou and Triantafyllou, 1995] was one of the first examples to propel itself successfully. Similar to a real tuna, it wasn't very maneuverable, but the robot proved the superior efficiency to other types of UAVs. Then, anguilliform-type swimming agile robots have been developed [Boyer et al., 2009, Boyer et al., 2008, Boyer et al., 2010, Kelasidi et al., 2016, Kelasidi et al., 2018] with several internal motors that imitate limbs of undulation. In the last decade, many robotic fish prototypes have been developed. Most of the biomimetic robotic fish are single-joint (one actuator) [Zhu et al., 2019, Yu et al., 2016, Clapham and Hu, 2014a, Clapham and Hu, 2014b], multi-joint (multiple actuators) [Li et al., 2021a, Martins et al., 2017, Yu et al., 2014b] or using smart materials [Gao et al., 2011, Currier et al., 2020, Katzschmann et al., 2018] to undulate its tail. One of these fish using hydraulics [Katzschmann et al., 2018] with silicon-based tail for undulation, can also control the depth of immersion via pump-based buoyancy mechanism. To achieve biomimetics, many robotic fish use soft materials for the rear body and caudal fin actuated by several wires [Floryan et al., 2018, Berg et al., 2021, Mitin and Lobov, 2022, Du et al., 2015, Zhong et al., 2017]. It has been shown that a biomimetic robotic tuna fish can closely resemble the real tuna and exhibit similar characteristics in undulation frequency and COT [1] calculations [Zhu et al., 2019].

1.3 Control methods for steering a robotic fish

Traditional control approaches of a robotic fish often rely on predefined kinematic patterns or manually-tuned control algorithms. These methods, while effective to a certain degree, do not necessarily harness the full potential of fluid-robot interactions. One of the common methods is to

1. Cost of Transport, the energy necessary to travel a certain distance

generate artificial central pattern generators. Central Pattern Generators (CPG) exist in many fish species. CPG is a biological neural network capable of producing coordinated rhythmic patterns without any rhythmic inputs from sensory feedback or central control system [Ijspeert, 2008]. Some research work use CPG [Yu et al., 2014b,Wang et al., 2019,Yu et al., 2014a] to propel robotic fish, while other work uses fuzzy logic [Hu et al., 2009a] to follow the target, PID controllers [Barbera et al., 2011] to regulate the orientation and a proprioceptive mechanism to propel the robotic fish [Sánchez-Rodríguez et al., 2021].

The control of a robotic fish in a turbulent flow is inherently not simple, due to the turbulent nature of the environment. Fish have evolutionarily developed a sense of proprioception in water. The sense of proprioception is a feeling of position of oneself with respect to the complex external environment. To feel the water flow, fish have developed Lateral Line System (LLS) [Coombs and Montgomery, 2014] that permits them to sense the flow field. The LLS system can be artificially mimicked by several pressure sensors on each side of a fish [Zheng et al., 2021]. Because of the complex nonlinearity of a robotic fish swimming, recently, there has been a growing interest in applying Reinforcement Learning (RL) for different control tasks [Zheng et al., 2021, Chen et al., 2022, Zhang et al., 2020, Rajendran and Zhang, 2022]. Most of this research work had predefined patterns (harmonic movement or CPG) and optimized some parameters of the predefined control pattern via Reinforcement Learning. In all previous work, there has not been a clear optimization on the best control strategy to achieve the highest swimming speed. We expand on this challenge in Chapter 3.

Machine learning (ML) has transformed different industries, offering tools to derive complex patterns and relationships from data that are often beyond human intuition. In the domain of robotics, ML represents a paradigm shift : rather than programming robots, we can now make them learn useful control techniques. RL, a subset of ML, is a control framework that allows robots to learn from trial and error, fine-tuning control strategies based on the feedback from the environment. For robotic fish, this means the potential to learn optimal swimming patterns, adapt to changing environmental conditions, and even develop novel maneuvers that might not have been conceived through traditional design approaches.

Lately, some research has studied fish swarm control, examining both collective behavior and the influence of individual fish on the swarm [Berlinger et al., 2021, Zhang et al., 2021, Zhang et al., 2017, Li et al., 2021b]. Conventional control algorithms may be inefficient for this control task and RL, especially multi-agent RL [Buşoniu et al., 2010] have a big potential to bring insights on fish swarm formation and movement. Moreover, different vision-recognition applications with a robotic fish have been explored [Ji et al., 2020, Angani et al., 2020, Hu et al., 2009b] with an aim to recognize objects, hand gestures and a target of interest.

This thesis aims at finding suitable tools via modelling and machine learning to find optimal swimming gaits of fish locomotion. Fish swimming capabilities result from years of evolution, and we will try to mimic fish swimming capacity by learning techniques on a biomimetic robotic fish described in Chapter 3. One of the ways to address the problem is to apply learning algorithms of the control theory on a biomimetic robotic fish to infer the optimized swimming gait. Lately, the work on fish locomotion gave some new insights on comprehension of fish movement [Sánchez-Rodríguez et al., 2020, Sánchez-Rodríguez et al., 2021]. These models are extensively used in simulation of a robotic fish in Chapter 3 in order to test the feasibility of the learning approach before proceeding with the real experimental setup. Then, we use the learning algorithms directly on biomimetic robotic fish to study an optimal swimming gait in experiments. Application of deep reinforcement learning approaches on a robotic fish locomotion and combination of these

algorithms with the models in simulations allows us to find optimal physical parameters A, ω (A being the amplitude of fin undulation and ω the angular frequency) of the fish kinematics for different tasks. The RL algorithms can potentially enhance the agility, efficiency and adaptability of biomimetic robotic fish, bringing them closer to their biological counterparts in performance. We will elaborate on the details of various RL algorithms in Chapter 2 and 3, and in the next section, the general introduction of machine learning is given.

1.4 Machine learning

Machine learning is a subset of Artificial Intelligence (AI) that provides algorithms to automatically infer meaningful patterns from provided or generated data. Machine learning yields insights about the data and recognizes complex relations among the data components. It has many applications in daily life, starting with chat-bots with general information assistance and ending with algorithms that pilot cars and satellite systems. The first appearances of machine learning happened in the 1950s with a "perceptron" [Rosenblatt, 1958] and a program learning to play checkers [Samuel, 1959]. Perceptron is the basic component of modern neural networks. After a few decades, the concept of Convolutional Neural Network (CNN) for pattern recognition was introduced in 1980 [Fukushima, 1980]. Later, the term deep learning was introduced in [Dechter, 1986] referring to the neural network with multiple hidden layers. The foundations of deep learning has been laid in the 1990s [Hinton et al., 1990, Hinton et al., 1984, Lecun et al., 1998]. The domain of deep learning has not been in the sight of the wide public until 2012 when the neural network AlexNet [Krizhevsky et al., 2012] was introduced. This artificial neural network (ANN) was the first instance of deep learning using CNNs which was able to decently classify the objects. Then the revolutionizing work in the domain of deep learning in the 2010s has been an introduction of GAN [Goodfellow et al., 2014] and Transformer [Vaswani et al., 2017] laying the foundation of Generative AI. The domains of engineering impacted the most by machine learning for today are Computer Vision and Natural Language Processing. Every industry has seen some direct or indirect impact created by the AI boom. The most recent breakthrough advances in the application of deep learning include AlphaFold [Jumper et al., 2021] predicting the architecture of protein structures, AlphaGo [Silver et al., 2016] learning to play the game "GO" and large language models (LLM) [Touvron et al., 2023] such as ChatGPT.

There are 3 types of machine learning with their sub-variants and mixtures Fig. 1.5 :

1. *Supervised Learning* : algorithms are trained on labeled data. Trained algorithms generalize and predict labels accurately on unseen data.
2. *Unsupervised Learning* : algorithms are used to infer a function and to describe hidden structures from unlabeled data.
3. *Reinforcement Learning* : algorithms are trained on self-generated data where algorithms improve in a particular task based on some reward or punishment.

There are several mixtures of supervised and unsupervised learning such as self-supervised learning, semi-supervised learning and weakly-supervised learning. Among the three types of machine learning, a particularly exciting approach is Reinforcement Learning (RL).

Reinforcement Learning, being a type of Machine learning, is a branch of control theory that focuses on the control of dynamical systems from system interaction and without prior modelization. Deep RL, referring to RL with neural nets, is considered as one of the promising venues of

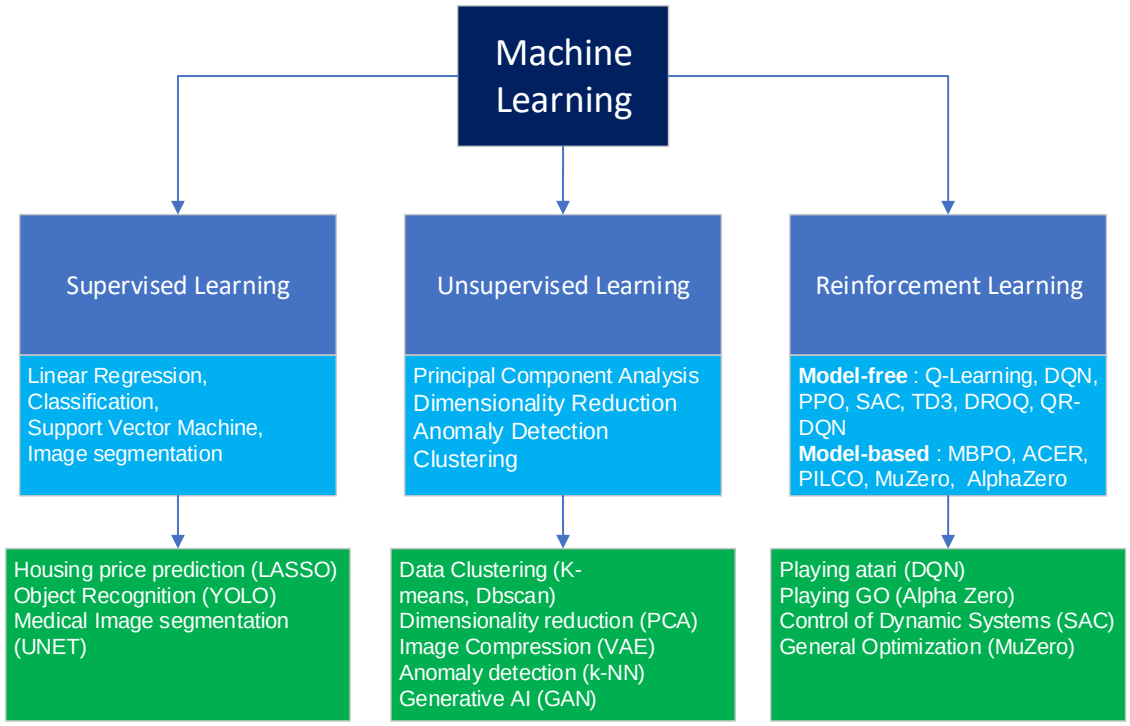

Figure 1.5 – Machine learning overview with light blue rectangles as model examples and green blocks as application examples.

AI. It has historically been applied for finding successful strategies in games [Tesauro, 1995, Mnih et al., 2013, Silver et al., 2017]. Then, deep RL has also shown enormous success in the control of dynamic systems both in simulation [Mnih et al., 2013, Lillicrap et al., 2016] and real robots [Kim et al., 2004, Ng et al., 2006, Buşoniu et al., 2018, Riedmiller, 2005].

There are several sub-types of deep RL :

1. Model-free : algorithms concentrate only on the actions leading to the desired outcome. Based on the interaction with a dynamical system, these methods model the control strategies. This subtype was a primary focus of algorithms applied in this thesis.
2. Model-based : algorithms not only optimize the control sequence to achieve the desired outcome, but also model the dynamics of the system from interaction experience via function approximations such as neural networks. This permits the algorithm to achieve the desired result with less system solicitation. Unlike model-based RL, model-free family do not model the physical system or the dynamics inner working leading to the computation superiority of model-free algorithms and potential better performance due to the "model-bias".

Before implementing data-driven learning algorithms on a robotic fish, we use reinforcement learning on the inverted pendulum problem in Chapter 2, usually seen as a benchmark problem in the control theory, because fish locomotion is a more difficult problem which involves fish body and complex fluid-structure dynamics defined by allometric laws [Pedley, 1977, Vogel, 2020, Gibouin et al., 2018, Sánchez-Rodríguez et al., 2020]. There is no guarantee that RL will be able to find good policies, as aquatic locomotion remains a highly nonlinear process.

1.5 Thesis structure

The present thesis is organized in 3 chapters.

1. **Context, Thesis structure**. The preset chapter details the context of PhD, aquatic and fish locomotion. The use cases of a robotic fish are presented with a gentle introduction and motivation to AI. Different types of robotic fish and various control strategies are introduced.
2. **Control theory methods with an application on an inverted pendulum**. This chapter details the general theory on control algorithms. These methods are then applied on an inverted pendulum including classical control theory methods. Extensive theory on data-driven control algorithms is given with an application on a cartpole system.
3. **Reinforcement learning approach to control a robotic fish**. This chapter details the materials and methods for the optimization of a swimming gait of a robotic fish. The modelization of a robotic fish is presented with the corresponding simulation model. First the optimization happens on data from a force sensor, then using camera readings. Our learning sessions targeted two main objectives : the maximization of thrust force with a stationary fish and the maximization of speed with a fish in motion within a water tank.

Research output

1. Israilov, S., Fu, L., Sánchez-Rodríguez, J., Fusco, F., Allibert, G., Raufaste, C., and Argentina, M. (2023). Reinforcement learning approach to control an inverted pendulum : A general framework for educational purposes. Plos one, 18(2) :e0280071.
2. Israilov, S., Fu, L., C. Brouzet, Allibert, G., Raufaste, C., and Argentina, M. Bio-inspired robotic fish thrust optimization using machine learning. Euromech Colloquium 628, Poster Session on "Complex particles in turbulent flow". May 4, 2023, Auditorium of the ICM, Nice, France.
3. Fu, L., Israilov, S., Brouzet, C. , Allibert, G., Raufaste, C., and Argentina, M. Optimum control strategies for maximum thrust production in underwater undulatory swimming. Submitted to Nature Communications (reference number : NCOMMS-23-57619)
4. Israilov et al. Learning to swim with deep Reinforcement Learning. *In process of writing*. IROS/RAL 2024

CHAPTER 2

Control theory methods with an application on an inverted pendulum

Controlling a nonlinear systems may be challenging. For this reason, the model-based control theory methods are explained with an application on the inverted pendulum, before moving with the data-driven machine learning approaches for control. Model-based control theory methods serve as an introduction and benchmark methods for control of data-driven reinforcement learning methods. We, hereby, provide basic tools to understand reinforcement learning algorithms. Based on simple principles, we expand on the model-free deep reinforcement learning enhanced with neural networks. Finally, we perform the analysis of the application described methods on a real pendulum setup and virtual environment simulators.

2.1 Traditional control theory methods

Control systems are an integral part of modern technology, guaranteeing stability and precision in areas ranging from industrial processes to robotics and aerospace. Many physical systems are inherently nonlinear and described by complex differential equations. However, they can be linearized around an equilibrium. After linearization, these systems can be modeled with several coupled first-order ODEs. Classical control theory has developed a number of methods that deal with these linearized systems.

Let the system have a state $\mathbf{X} = [x_1, x_2 ... x_n] \in \mathbb{R}^n$, output $\mathbf{Y} = [y_1, y_2 ... y_p] \in \mathbb{R}^p$ and control $\mathbf{U} = [u_1, u_2 ... u_m] \in \mathbb{R}^m$ applied on the state $\mathbf{X}$, then a time-invariant system characterized by nonlinear dynamics can be generally expressed through a set of nonlinear differential equations in the following form :

$$\begin{cases} \dot{\mathbf{X}}(t) = f(\mathbf{X}(t), \mathbf{U}(t)) \\ \mathbf{Y}(t) = g(\mathbf{X}(t)), \end{cases} \tag{2.1}$$

where f, g are locally Lipschitz (continuous) nonlinear vector functions. We consider that there is no direct relation between input $\mathbf{U}$ and output $\mathbf{Y}$. Around the equilibrium point $\mathbf{X}^*$ when $f(\mathbf{X}^*, 0) = 0$, the dynamics of this nonlinear system $f(\mathbf{X}(t), \mathbf{U}(t))$ can be written in the following form (via Taylor expansion) :

$$\dot{\mathbf{X}} = \left(\frac{\partial f}{\partial \mathbf{X}}\right)_{\mathbf{X}=\mathbf{X}^*, \mathbf{U}=0} \mathbf{X} + \left(\frac{\partial f}{\partial \mathbf{U}}\right)_{\mathbf{X}=\mathbf{X}^*, \mathbf{U}=0} \mathbf{U} + f_{h.o.t.}(\mathbf{X}, \mathbf{U}), \tag{2.2}$$

where $f_{h.o.t.}(\mathbf{X}, \mathbf{U})$ denotes the higher-order terms in $\mathbf{X}$ and $\mathbf{U}$. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ represent the Jacobian matrix of f with respect to $\mathbf{X}$, evaluated at the point $(\mathbf{X} = \mathbf{X}^*, \mathbf{U} = 0)$; similarly, let $\mathbf{B} \in \mathbb{R}^{n \times m}$ be the Jacobian matrix of f with respect to $\mathbf{U}$ and $\mathbf{C} \in \mathbb{R}^{n \times p}$ denote the Jacobian matrix of g with respect to $\mathbf{X}$ evaluated at $\mathbf{X}^*$. Then, near equilibrium $\mathbf{X}^*$ such that $f(\mathbf{X}^*, \mathbf{U})) = 0$, we can approximate this non-linear system with the Linear Time-Invariant (**LTI**) system by neglecting the higher-order terms. The system can be modeled with the **state-space equation** formalism as :

$$\begin{cases} \dot{\mathbf{X}}(t) = \mathbf{A}\mathbf{X}(t) + \mathbf{B}\mathbf{U}(t) \\ \mathbf{Y}(t) = \mathbf{C}\mathbf{X}(t), \end{cases} \tag{2.3}$$

where $\mathbf{A}, \mathbf{B}$ are the state and control matrices respectively. The matrices $\mathbf{A}, \mathbf{B}$ describe the dynamics and the coupling of control with the dynamical system, respectively. $\mathbf{C}$ is an output matrix that define how the state of the system affects the outputs of this system. This formalism is defined for continuous-time systems and can further be adapted for discrete-time systems.

We proceed with an illustrative example of the linearization of a simple pendulum in Fig. 2.1 defined by the unit mass m, the unit length l, viscous friction coefficient k_v and actuated by the torque T counterclockwise along the pendulum axis. According to Newton's second law, the actuated pendulum can be modeled with the second-order ODE as :

$$\ddot{\theta} = \frac{T}{ml^2} - \frac{g}{l}\sin\theta - \frac{k_v}{m}\dot{\theta}. \tag{2.4}$$

We choose to linearize a system around its unstable equilibrium $\theta^* = \pi$, where θ is an angle of a pendulum. Around this fixed point ($\theta^* = \pi$), $\sin\theta \approx \theta$ in a small angle limit. Considering θ to be the angular position and ω the angular velocity of the pendulum, let the state of this system

be defined with $\mathbf{X} = [\theta, \omega]$; then the system dynamics with actuation near $\theta^* = \pi$ can be modeled with the following pair of differential equations :

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = \frac{T}{ml^2} - \frac{g}{l}\theta - \frac{k_v}{m}\dot{\theta} \end{cases} \tag{2.5}$$

or in a continuous state-space equation :

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & -\frac{k_v}{m} \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} \theta \\ \omega \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix}}_{\mathbf{B}} T \tag{2.6}$$

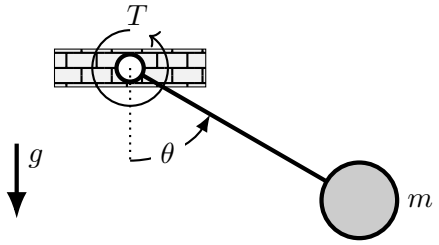

Figure 2.1 – Sketch of the inverted pendulum.

The outputs of the system are the state values of interest or the measured state variables. For instance, if only an angular position θ is extracted and used for control, then the output vector would be defined as :

$$\begin{bmatrix} \theta \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{\mathbf{C}} \begin{bmatrix} \theta \\ \omega \end{bmatrix} \tag{2.7}$$

For the following sections, the simple pendulum example will be used to illustrate the control theory concepts until we model the real inverted pendulum system named cart-pole and apply the described control methods on it.

Most of the control techniques in practice require the feedback for a viable control to the reference state $\mathbf{X}_c(t)$ called "setpoint" and become "closed-loop" systems. In the following sections, PID (Proportional Integral Derivative) and LQR (Linear Quadratic Regulator) are presented to drive a linearized system to its equilibrium. Then, we will extend on the Lyapunov theory, providing the necessary tools to drive nonlinear systems to equilibrium under certain conditions. Later in this thesis, these techniques will be applied on the real inverted pendulum system.

2.1.1 PID

A Proportional Integral Derivative (PID) controller [Kailath, 1980] is a state feedback control mechanism that is designed to drive a dynamical system to a pre-defined setpoint or trajectory for a given measured state variable $\mathbf{Y}(t)$. Due to its simplicity and easiness of tuning, it is widely adopted in every industry. In addition, it does not require the knowledge of model dynamics for

its functioning. The control mechanism consists of minimization of error along some state dimension : $e(t) = \mathbf{Y}(t) - \mathbf{Y}_c(t)$, which is an error between the actual state variable of the system and the desired state variable. The PID control consists of three terms :

$$\mathbf{U}(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}, \tag{2.8}$$

where :

- $\mathbf{U}(t)$ is the controller output
- K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively
- **Proportional term :** $K_p e(t)$ is responsible for the immediate error correction. If the error is large, the control output of the proportional term is big as well, and vice versa. The term by itself can not bring the system to the setpoint as it incorporates the static steady-state error. []
- **Integral term :** $\int_0^t e(\tau)d\tau$ is responsible for static error correction. It integrates the error through time and adjusts the response accordingly. Just *PI* controller is sufficient to control lots of dynamical systems.
- **Derivative term :** $K_d \frac{de(t)}{dt}$ is responsible for changing trend anticipation and adaptation. As it is calculating the derivative of an error, it foresees when the error will become zero and adapts the response accordingly. Compared to *PI* controller, *D* term in PID controller helps with faster convergence to the setpoint and dumps the overshoot of *PI* components. In many cases in practice, it is difficult to implement a derivative term of a PID controller, as the derivative of a noisy signal is amplifying the noise and may diverge. Low-pass filtering [Oppenheim et al., 1999] or Kalman [Kalman, 1960] filtering is a common practice when dealing with the derivative component of a PID controller.

The tuned PID controller balances these 3 components to achieve the desired state of a physical system under certain constraints.

There are many different tuning mechanisms for the PID controller : Ziegler–Nichols [Ziegler and Nichols, 1942], Cohen-Coon method [Cohen and Coon, 1953], genetic algorithms [Chen et al., 2013] etc.

In many cases, the PID controller is sub-optimal, where an optimal controller refers to the controller that controls a system with optimized objective in the best possible way. While PID is very popular, there are many more advanced control strategies that are more perform-ant and robust to noise. PID usually regulates one component of the state and can not tackle complex objectives. To address these challenges, in the following section, we describe the optimal model-based control algorithm for linearized systems.

2.1.2 Linear Quadratic Regulator (LQR)

LQR is a method of control theory that regulates a linear system in an optimal way under given constraints [Kailath, 1980]. It optimizes the quadratic cost and finds the state feedback control. It takes into account the dynamics of the system unlike PID, and not only it drives the system to the desired setpoint, but it also minimizes the amount of control taken to get there. Given an LTI system with the state-space representation given as $\dot{\mathbf{X}}(t) = \mathbf{A}\mathbf{X}(t) + \mathbf{B}\mathbf{U}(t)$, LQR is the optimal controller which minimizes the infinite time horizon quadratic cost functional defined by :

$$J = \int_0^\infty \left(\mathbf{E}_X(t)^\top \mathbf{Q}\mathbf{E}_X(t) + \mathbf{U}(\mathbf{t})^\top \mathbf{R}\mathbf{U}(\mathbf{t})\right) dt, \tag{2.9}$$

wherein $\mathbf{E}_X(t) = \mathbf{X}(t) - \mathbf{X}^\star$ represents the error between a system's state $\mathbf{X}$ and the desired setpoint $\mathbf{X}^\star$. The weighing matrices $\mathbf{Q} = \mathbf{Q}^\top \geq 0$ and $\mathbf{R} = \mathbf{R}^\top \geq 0$, which are generally diagonal, can be used as design parameters to penalize state errors and the control signal. They indicate the relative importance of state regulation to the desired setpoint with control effort. In general, weights in $\mathbf{Q}$ are greater than in $\mathbf{R}$, as the state regulation is more important than the action effort taken to get to the state.

It can be shown that the solution to the optimal control problem above is a linear state feedback in the form :

$$\mathbf{U}(t) = -\mathbf{K}\mathbf{E}_X(t), \tag{2.10}$$

where $\mathbf{K}$ is a constant gain matrix obtained from

$$\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^\top\mathbf{P} \tag{2.11}$$

and $\mathbf{P}$ is the solution of the continuous-time Algebraic Riccati Equation

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} + \mathbf{Q} = 0. \tag{2.12}$$

LQR can also be adapted to discrete-time dynamical systems.

As an example, the performances of LQR and PID controllers were investigated in a discrete simulation (sampling time $\Delta t = 0.01s$) of an inverted pendulum. The simulation uses Eq. (2.4), with the goal of pendulum stabilization at the unstable equilibrium $\theta^* = \pi$. The system has neither viscous friction nor white noise on $\mathbf{X}, \mathbf{U}$ as it occurs in practice. Starting from the initial state : $\mathbf{X_0} = [\pi + 0.5, 0]$, numerical simulation of a simple pendulum is done using an iterative scheme with Runge-Kutta integration precise to the second order :

$$\begin{cases} \dot{\theta}_{t+1} = \dot{\theta}_t + \Delta t(u_t - \frac{g}{l}\sin\theta_t) \\ \theta_{t+1} = \theta_t + \Delta t\dot{\theta}_t + \frac{1}{2}\Delta t^2(u_t - \frac{g}{l}\sin\theta_t), \end{cases} \tag{2.13}$$

where u_t is a control action computed at each time step with $u_{LQR}(t) = -\mathbf{K}\mathbf{E}_X(t)$ for the discrete LQR and $u_{PID}(t) = K_p e_{\theta_t} + K_i \int_0^t e_\theta(\tau)d\tau + K_d\dot{e}_{\theta_t}$ for the PID controller. The simulation results are illustrated in Fig. 2.2.

The methods presented so far are valid for linear systems. In many cases of nonlinear systems, such as a swing-up of the pendulum, it is not possible to control the system with PID or LQR controllers. To tackles this challenge, we present the theory that analyses the stability of a nonlinear system and provides the necessary means to control the system to a desired equilibrium.

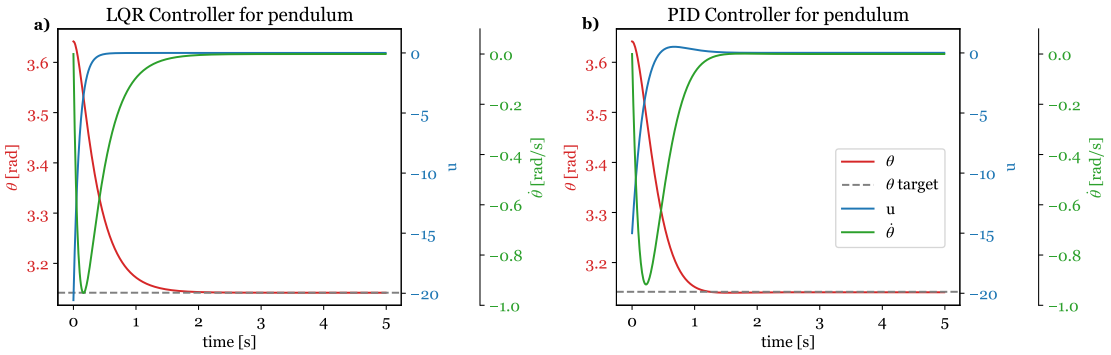

Figure 2.2 – Illustration of LQR and PID controllers with a) Discrete-time LQR parameterized by $\mathbf{Q} = diag([10, 1])$ and $\mathbf{R} = diag([0.01])$ in their diagonal entries, implying that optimization happens on θ. The $\mathbf{K}$ matrix gain found from solving Ricatti equation is $[-40.52, -13]$. As we can see, the controller is capable of driving θ to the desired setpoint, and the controller actuation is strong at the beginning. b) PID with $K_p = 30, K_i = 0.1, K_d = 8$. It is also capable of stabilizing the pendulum. The overshoot on the action corresponds to the reversing of applied torque with the anticipation of the target θ and slowing down the pendulum.

2.1.3 Lyapunov theory

Stability theory has a key role in systems theory and control engineering. Hereby, we are interested in the stability of equilibrium points. For the LTI systems in Eq. (2.6), stability of the equilibrium point [1] is defined by the negativeness of the real part of eigenvalues of the matrix $\mathbf{A}$ and $(\mathbf{A} - \mathbf{BK})$ with the control defined as $\mathbf{U}(t) = -\mathbf{K}\mathbf{X}(t)$. Stability of nonlinear systems is more complicated as their evolution is not always predictable. The common tools to analyze the stability of nonlinear systems are linearization, bifurcation analysis [Kuznetsov et al., 1998] and Lyapunov theory [Lyapunov, 1892, Mawhin, 2005, Åström and Furuta, 2000, Durand et al., 2013].

Lyapunov theory is the most general method to analyze the stability of nonlinear systems. The stability of a system at equilibrium points is determined using *Lyapunov function* that is always positive and decreases along the trajectories of a system. Fulfilling certain conditions, the existence of such a function proves the local stability of the system. The theory is named after the Russian mathematician and engineer who pioneered the theory.

A nonlinear dynamical system can be represented by state-space equation :

$$\dot{\mathbf{X}}(t) = f(\mathbf{X}(t), \mathbf{U}(t)), \tag{2.14}$$

where $f(\mathbf{X}(t), \mathbf{U}(t))$ is a locally Lipschitz (continuous) nonlinear function. Lyapunov theory analyzes the stability and convergence of a nonlinear system at equilibrium points. For the sake of simplicity, we omit the time dependance notation in $\mathbf{U}(t) \to \mathbf{U}, \mathbf{X}(t) \to \mathbf{X}$, for the rest of this section.

Lyapunov theory : Let $\mathbf{X}^*$ be an equilibrium point of Eq. (2.14) and $\Omega \subset \mathbb{R}^n$ be the set of available states of the system containing $\mathbf{X}^*$. Let candidate Lyapunov function $V(\mathbf{X}) : \Omega \to \mathbb{R}$ be a continuously differentiable function such that :

1. equilibrium point $\mathbf{X}^*$ for an LTI system means $\mathbf{A}\mathbf{X}^*(t) = 0$; for a nonlinear system, equilibrium point $\mathbf{X}^*$ implies $f(\mathbf{X}^*, 0) = 0$

1. $V(\mathbf{X}^*) = 0$
2. $V(\mathbf{X}) > 0$ for all $\mathbf{X} \in \Omega$ except $\mathbf{X}^*$
3. $\dot{V}(\mathbf{X}) = \sum_{i=0}^{n} \frac{\partial V}{\partial \mathbf{X}_i} \dot{\mathbf{X}}_i = \sum_{i=0}^{n} \frac{\partial V}{\partial \mathbf{X}_i} f_i(\mathbf{X}) < 0$ for all $\mathbf{X} \in \Omega$ except $\mathbf{X}^*$,

then the system is asymptotically stable in $\mathbf{X}^*$, meaning that all solutions starting at nearby points will eventually converge in $\mathbf{X}^*$ as time approaches infinity. If the derivative of Lyapunov function $\dot{V}(\mathbf{X}) \leq 0$, then the system is *Lyapunov* stable in $\mathbf{X}^*$, meaning that all solutions starting at nearby points will converge to oscillate at the equilibrium $\mathbf{X}^*$.

Lyapunov function is often related to the mechanical energy of a system due to its conservation properties. As an illustrative example, in the case of pendulum defined by Eq. (2.4) without viscous friction and control, we choose Lyapunov function to be the energy of the pendulum system :

$$V(\mathbf{X}) = \left(\frac{g}{l}\right)(1 - \cos\theta) + \frac{1}{2}\omega^2. \tag{2.15}$$

According to Eq. (2.5) without viscous friction and control, $\dot{V}(\mathbf{X})$ is derived as :

$$\begin{aligned} \dot{V}(\mathbf{X}) &= \left(\frac{g}{l}\right)\omega\sin\theta + \omega\dot{\omega} \\ &= \left(\frac{g}{l}\right)\omega\sin\theta - \left(\frac{g}{l}\right)\omega\sin\theta = 0, \end{aligned} \tag{2.16}$$

which concludes that the pendulum is Lyapunov stable. This is rationalized due to the lack of viscous friction and conservation of energy. If there is an energy dissipation due to viscous friction, as in Eq. (2.4), then this Lyapunov function yields :

$$\begin{aligned} V(\mathbf{X}) &= \left(\frac{g}{l}\right)(1 - \cos\theta) + \frac{1}{2}\omega^2 \\ \dot{V}(\mathbf{X}) &= \left(\frac{g}{l}\right)\omega\sin\theta + \omega\dot{\omega} \ = -(k_v)\,\omega^2, \end{aligned}$$

which means that Lyapunov candidate function demonstrates that energy $V(\mathbf{X})$ will not increase, but fails to describe the asymptotic stability of the damped pendulum. The derivative $\dot{V}(\mathbf{X}) = 0$, when $\omega = 0$ which happens at the maximum of the potential energy, but the pendulum is not at rest as $\ddot{\theta} \neq 0$, unless the pendulum is at $\mathbf{X}^* = [0, 0]^T$, from which it can be inferred that the pendulum is asymptotically stable at $\mathbf{X}^* = [0, 0]^T$.

We can also define another Lyapunov function (for simplicity of notation, we note $\frac{g}{l} = a$ and we choose $b = \frac{k_v}{m} > 0$) :

$$V(\mathbf{X}) = \frac{1}{2}\mathbf{X}^T P \mathbf{X} + a\,(1 - \cos\theta) = \frac{1}{2}\begin{bmatrix} \theta & \omega \end{bmatrix}\begin{bmatrix} \frac{b^2}{2} & \frac{b}{2} \\ \frac{b}{2} & 1 \end{bmatrix}\begin{bmatrix} \theta \\ \omega \end{bmatrix} + a\,(1 - \cos\theta)$$

and its derivative given by :

$$\begin{aligned} \dot{V}(\mathbf{X}) &= \left(\frac{b^2}{2}\theta + \frac{b}{2}\omega + a\sin\theta\right)\omega + \left(\frac{b}{2}\theta + \omega\right)(-a\sin\theta - b\omega) \\ &= a\,(1-1)\,\omega\sin\theta - a\frac{b}{2}\theta\sin\theta + \left(\frac{b^2}{2} - \frac{b}{2}b\right)\theta\omega + \left(\frac{b}{2} - b\right)\omega^2, \\ &\text{implying} \quad \dot{V}(\mathbf{X}) = -\frac{1}{2}ab\theta\sin\theta - \frac{1}{2}b\omega^2. \end{aligned}$$

Having the domain of available states as $\mathbf{X} \in \Omega = \mathbb{R}^2$ with $|\theta| < \pi$ and matrix P being positive definite such as $\frac{1}{2}\mathbf{X}^T P\mathbf{X} > 0$ implies that $\forall \mathbf{X} \neq \mathbf{X}^*, V(\mathbf{X}) > 0$ and $\dot{V}(\mathbf{X}) < 0$ guaranteeing asymptotic stability of a pendulum at $\mathbf{X}^* = [0, 0]^T$.

Controller based on the Lyapunov theory consists in injecting energy into the system until it reaches its equilibrium. Its stability analysis is defined by *Control-Lyapunov Function*. For a system with given dynamics $\dot{\mathbf{X}} = f(\mathbf{X}, \mathbf{U})$ and Lyapunov Function $V(\mathbf{X}, \mathbf{U}) > 0$:

$$\forall \mathbf{X} \neq 0, \exists \mathbf{U} \quad \dot{V}(\mathbf{X}, \mathbf{U}) = \frac{\partial V}{\partial \mathbf{X}} f(\mathbf{X}, \mathbf{U}) < 0 \quad \text{and} \quad \exists \mathbf{U} \quad \dot{V}(0, \mathbf{U}) = 0.$$

This implies that there is a control $\mathbf{U}$ that would allow to decrease V until $\mathbf{X} = 0$.

A simple pendulum example was given to demonstrate the concepts of control theory in illustrations. In the next section, Lyapunov theory based controller will be investigated and applied on a real inverted pendulum with the cart to swing-up the pendulum to its unstable equilibrium.

2.2 Controlling the pendulum using model-based techniques

2.2.1 Introduction

Inverted pendulums - also known as “cart-pole” apparatuses - belong to a simple type of system that have a long history in the field of mechanics and dynamical systems [Lundberg and Barton, 2010, Boubaker, 2013]. Their dynamics is described by a set of mathematical equations that are rather simple to derive, while still featuring interesting properties such as nonlinearity and under-actuation. This makes an inverted pendulum a perfect candidate to benchmark and showcase new control algorithms before deploying them on more complex systems such as quadrotors or humanoid robots [Sugihara et al., 2002]. In addition, given the simplicity required to build an experimental prototype, cart-pole systems are very well-suited for teaching a wide variety of topics, ranging from Lagrangian mechanics to control theory. Indeed, the literature includes numerous examples of low-cost pendulums designed and built with the purpose of teaching one or more subjects to undergraduates [Lee and Jung, 2008, Lazarini et al., 2014, Bakaráč et al., 2017].

In this section, we expand existing pedagogical works by providing a complete and multidisciplinary discussion that touches several relevant subjects, ranging from mathematical modeling of dynamical systems via the Lagrangian approach to nonlinear control theory using energy-based approaches and optimal linear control. Later in the thesis, advanced and recent algorithms coming from the Reinforcement Learning (RL) literature are presented and applied on this system. The reader is thus guided across all the steps required to understand, model and control such type of system, using either model-based or data-driven approaches. Moreover, the theoretical discussion presented hereby is accompanied by an open-source code repository which allows to replicate all the approaches presented here [Israilov et al., 2023]. It includes detailed instructions to build the prototype used in this work, configure its software interface and implement several controllers. We believe that this chapter and its complementary material can be a great resource for teachers that are willing to provide students with a strong and varied theoretical background, while also granting them the possibility to safely experiment with newly learned topics. The discussion is detailed enough to serve both as an inspiration for the teacher and as a tutorial for the student.

First of all, we derive the equations of the cart-pole system and we reduce them to a simpler, dimensionless model. In Section 2.2, we then apply LQR and Lyapunov theory techniques presented before to control the cart-pendulum which belong to the general class of model-based control techniques, since they require the equations of motion to be explicitly available.

2.2.2 Experimental setup and methods

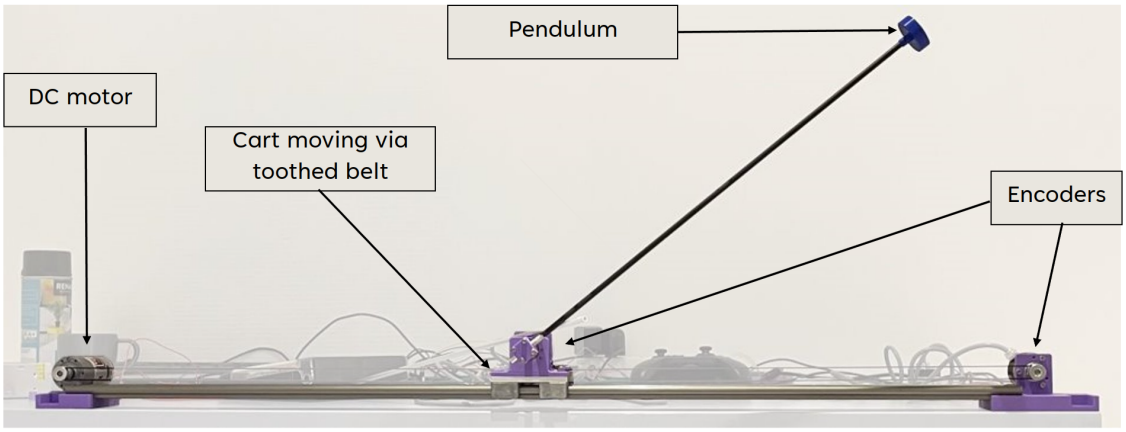

Figure 2.3 – The experimental setup.

The experimental realization of the pendulum is shown in fig.2.3. It features a DC motor (model : MFA 970D 12V) which can apply a horizontal force to the sliding base thanks to a transmission belt.

An incremental encoder measures the position x of the base on a linear track, assuming that $x = 0\,\mathrm{m}$ corresponds to the centered position. The finite length of the track gives the constraint $|x| < x_{max}$, with $x_{max} = 0.35\,\mathrm{m}$. A second encoder mounted on the moving base assesses the angle θ. Both are incremental encoders (model : LD3806-600BM-G5-24C) with two phases in quadrature, for a total of 2400 steps per revolution. A Raspberry Pi 4 is used to handle the electronic devices and control the system. It runs a C++ executable, namely the *low-level interface* (LLI), which is responsible of handling the different hardware components and exposes the current state of the pendulum to client control applications (Appendix A.1).The algorithm running on the Raspberry Pi actuates the motor, within the three possible actions. All the code to control the pendulum is open source and available, as well as a reference manual [Israilov et al., 2023].

2.2.3 Modeling the inverted pendulum and the controller

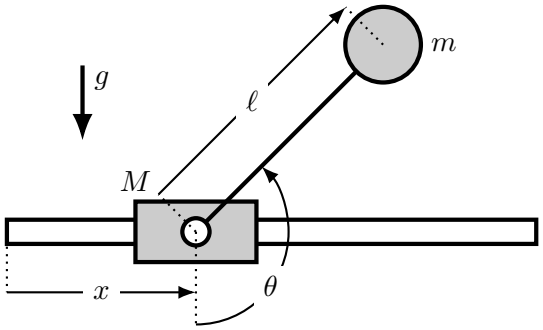

Figure 2.4 – Sketch of the inverted pendulum.

We assume a mass m located at the end of a massless rigid rod of length ℓ. Its other extremity is free to rotate on a motorized cart with mass M located at abscissa $x(t)$. The angle separating the rod to the downward vertical direction is $\theta(t)$, as shown in Fig. 2.4. The purpose is to stabilize the pendulum in its unstable equilibrium position $\theta = \pi$ by controlling the motion of the cart only, located at $x(t)$. In this section, we derive the equation that determines the dynamics of the angle $\theta(t)$, under the driving of the cart motion, using Lagrangian mechanics.

The position of the mass m is $(x(t) + \ell \sin\theta(t), -\ell\cos\theta(t))$. The Lagrangian of the system is $\mathcal{L} = K - W$, where K is the kinetic energy and W is the potential energy. For the sake of readability, in the following derivations, we will drop the temporal dependence (t) of physical variables.

According to Euler-Lagrange equation :

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = Q_i, \tag{2.17}$$

where Q_i being a generalized force and q_i is a "generalized coordinate". The Euler-Lagrange equation for an idealized (without friction) cart-pole system translates into :

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} &= 0 \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} &= 0. \end{aligned} \tag{2.18}$$

The Lagrangian of the system cart-pole writes in the following form :

$$\mathcal{L} = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m\left[\dot{x}^2 + \ell^2\dot{\theta}^2 + 2\ell\dot{x}\dot{\theta}\cos\theta\right] + mg\ell\cos\theta.$$

We calculate the different terms for the θ variable :

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta} &= -mg\ell\sin\theta - m\ell\dot{x}\dot{\theta}\sin\theta \\ \frac{\partial \mathcal{L}}{\partial \dot{\theta}} &= m\ell^2\dot{\theta} + m\ell\dot{x}\cos\theta \\ \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\theta}} &= m\ell^2\ddot{\theta} + m\ell\ddot{x}\cos\theta - m\ell\dot{x}\dot{\theta}\sin\theta. \end{aligned}$$

The Euler-Lagrange equation reads :

$$m\ell^2\ddot{\theta} + m\ell\ddot{x}\cos\theta + mg\ell\sin\theta = 0.$$

Thus, the Euler-Lagrange equation giving the temporal evolution of θ is :

$$\ddot{\theta} + \frac{1}{\ell}\left(g\sin\theta + \ddot{x}\cos\theta\right) = 0. \tag{2.19}$$

In what follows, we note $\omega = \sqrt{g/\ell}$ the natural frequency of the pendulum.

To close the system, it remains to write the equation describing the position of the motorized cart. Writing the Euler-Lagrange equation of the system for the x coordinate involves modeling the DC motor (Direct Current motor) dynamics with different types of frictions and mechanical coupling with the cart-pole system. Instead, we assume the following form :

$$\ddot{x} = \frac{1}{\tau}\left(\dot{x}_c - \dot{x}\right), \tag{2.20}$$

where $\dot{x}_c$ is the target velocity value provided by the controller, τ is the time scale taken by the cart velocity to converge to $\dot{x}_c$.

This model is quite idealized, and we thus reconsider to take into account other physical effects that influence the motion of the system. In particular, we consider viscous friction torque acting on the pivot joint of the pendulum and static friction resisting the motion of the base :

$$\ddot{\theta} + k_v\dot{\theta} + \frac{1}{\ell}\left(g\sin\theta + \ddot{x}\cos\theta\right) = 0, \tag{2.21}$$

$$\ddot{x} = \frac{1}{\tau}\left(\dot{x}_c - \dot{x}\right) - f_c\,\mathrm{sign}(\dot{x}) - f_d, \tag{2.22}$$

where k_v controls the amplitude of the viscous friction, while f_c and f_d are introduced to model asymmetric dry friction acting on the motorized base. Note that the effects of viscous friction acting on the cart have already been implicitly taken into account in eqn. 2.20. In experiments, the cart target velocity is proportional to the voltage U. Defining k_U the proportionality constant gives

$$\dot{x}_c = k_U U. \tag{2.23}$$

The cart is constrained to move on a track of length $2x_{max}$. The exact procedures to measure the physical parameters that appear in eqs. 2.21, 2.22, 2.23 are described in Appendix A.1. Their values are summarized in table A.1.

Since the system is governed by various physical parameters, we perform a dimensionless reduction technique to limit the number of controlling parameters. By using the scale changes $x = \ell p(t/T)$, and $\theta = \alpha(t/T)$ and choosing as a typical time scale $T = \sqrt{\ell/g}$, we obtain :

$$\frac{\ddot{\alpha}}{T^2} + \frac{k_v\dot{\alpha}}{T} + \frac{1}{\ell}\left(g\sin\alpha + \frac{\ell\ddot{p}}{T^2}\right) = 0, \tag{2.24}$$

$$\ddot{\alpha} + Tk_v\dot{\alpha} + \sin\alpha + \ddot{p}\cos\theta = 0, \tag{2.25}$$

$$\ddot{\alpha} + \nu\dot{\alpha} + \sin\alpha + \ddot{p}\cos\alpha = 0, \tag{2.26}$$

and

$$\frac{\ell\ddot{p}}{T^2} = \frac{\ell}{T\tau}(\dot{p}_c - \dot{p}) - f_c\,\mathrm{sign}(\dot{p}) - f_d \tag{2.27}$$

$$\ddot{p} = \mu\left(\dot{p}_c - \dot{p}\right) - \gamma_c\,\mathrm{sign}\,\dot{p} - \gamma_d, \tag{2.28}$$

where $\mu = T/\tau$ measures the ratio of the pendulum time scale to those of the motorized cart. The parameters $\gamma_c = f_c/g$ and $\gamma_d = f_d/g$ are directly related to the motorization of the cart and the possible incline of the track. The viscous drag of the pendulum, which is quite small in our experiments, is controlled by the parameter ν. The linear stability analysis of equation 2.26 performed around $\alpha = \pi$ with $\ddot{p} = 0$ shows a positive eigenvalue which renders the upward equilibrium unstable.

In the following, we neglect the presence of viscous friction at the pivot ($\nu \sim 0$) for the sake of simplicity.

2.2.4 Control results with model-based techniques

2.2.4.1 Lyapunov method for the swing-up phase

The idea of the swing-up method is to define a candidate Lyapunov function $V(\alpha, \dot{\alpha})$ whose value always decreases, *i.e.* , $\frac{dV}{dt} < 0$ and presents a global minimum at $\alpha = \pi$ and $\dot{\alpha} = 0$.

We define the mechanical energy of the pendulum by setting $\ddot{p} = 0$ in Eq. (2.26),

$$E(\alpha, \dot{\alpha}) = \frac{1}{2}\dot{\alpha}^2 - \cos\alpha. \tag{2.29}$$

For the swinging pendulum, we consider the mechanical energy to form a candidate for $V(\alpha, \dot{\alpha})$, which is standard in literature [Åström and Furuta, 2000, Durand et al., 2013]. In particular, the objective is to stabilize the pendulum at the upward position ($\alpha = \pi$) with null angular velocity ($\dot{\alpha} = 0$), wherein the energy is $E^\star = E(\pi, 0) = 1$. A candidate Lyapunov function can be thus written as :

$$V(\alpha, \dot{\alpha}) = \frac{1}{2}\left(E - E^\star\right)^2 = \frac{1}{2}\left(\frac{1}{2}\dot{\alpha}^2 - \cos\alpha - 1\right)^2, \tag{2.30}$$

According to Eq. (2.30) and Eq. (2.26), the temporal derivative of this quantity writes :

$$\dot{V} = \left(E - E^\star\right)\dot{\alpha}\left(\ddot{\alpha} + \sin\alpha\right) = -\left(E - E^\star\right)\ddot{p}\dot{\alpha}\cos\alpha. \tag{2.31}$$

Choosing the control :

$$\ddot{p} = k\left(E - E^\star\right)\dot{\alpha}\cos\alpha, \tag{2.32}$$

ensures that $\dot{V}$ is always negative. V is thus a Lyapunov function to swing up the pendulum and under the control action above, the system will converge towards $V = 0$ [see Fig. 2.5]. Note that the above defined Lyapunov function is related to the energy at the state $\mathbf{X} = [\alpha, \dot{\alpha}] = [\pm k\pi, 0]$, where $k \in \mathbb{R}$ and thus the convergence solution is not unique. Indeed, the solution of equation $V = 0$ corresponds not to a single point but to a trajectory in the phase space with a certain energy level. Hence, the controller with Lyapunov stability can be seen as a robust controller to drive the pendulum towards the stable manifold with the corresponding energy $E^\star = E(\pi, 0) = 1$.

We simulated the dynamics of the pendulum angle α subject to this Lyapunov control using a second-order Runge-Kutta integration scheme and sampling time $0.002\,\mathrm{s}$, as shown in Fig. 2.5. Near the unstable equilibrium, the lyapunov function is close to zero, thus the control of the system is very sensitive to the numerical errors, and this produces oscillations with a large period, in which the pendulum stays upright before falling down and getting back to $\alpha = \pm k\pi$, as seen in Fig. 2.5. The coefficient k in Eq. (2.32) is chosen to be 1.

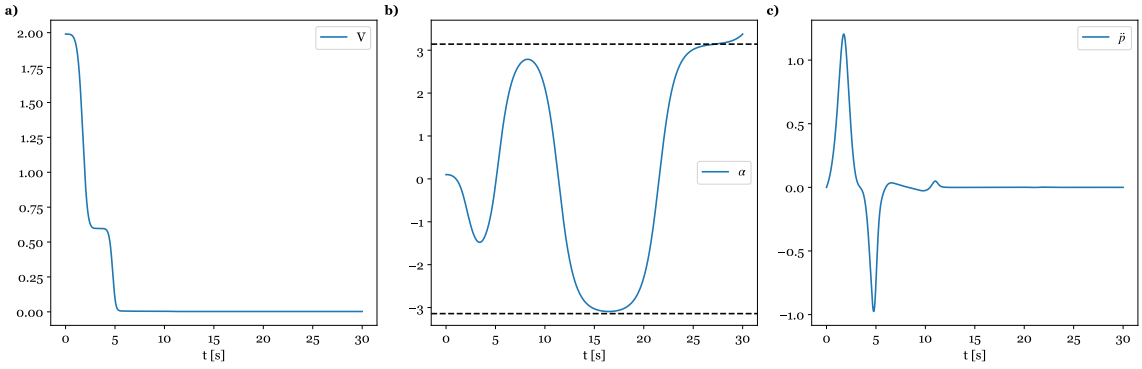

Figure 2.5 – Temporal evolution of the a) Lyapunov function V, b) the angle α and c) cart command $\ddot{p}$ for using the Lyapunov-theory based control technique.

Applying the Lyapunov-theory based controller on the motion of the cart will quickly drive the pendulum near its unstable equilibrium. In this configuration, the dynamical equation can be

linearized and the control will be notably simplified. In the next subsection, we design a linear controller to maintain a pendulum in its unstable equilibrium.

2.2.4.2 LQR for stabilization

The swing-up method detailed in the previous section is designed to control the angular behavior of the pendulum, while no action is taken to actively regulate the position of the base and stabilize the pendulum around its unstable equilibrium. To overcome this problem, we propose to use a Linear-Quadratic-Regulator (LQR) to stabilize the cart in its reference position (typically, the center of the rail) while maintaining the pendulum in the upright configuration. Note that, since the linearized model is valid only in a neighborhood of the upward configuration, the LQR strategy is applicable only when the swing-up phase has been already completed.

Near the unstable equilibrium, the dynamics are slow and linear and therefore it should be easier to control the system in this region. We linearize 2.26 around the unstable equilibrium $\alpha = \pi + \alpha_1, \dot{\alpha} = 0 + \dot{\alpha}_1$, where α_1 is small :

$$\ddot{\alpha}_1 - \alpha_1 - \ddot{p} = 0. \tag{2.33}$$

As expected, when $\ddot{p} = 0$ the upward vertical position is an unstable equilibrium point for the system. The purpose of the linear control will be to derive an equation for $\ddot{p}$. The driving velocity $\dot{p}_c$ will be found by solving the p equation from 2.28. A simple way to accomplish the control is to set $\ddot{p}$ to be a linear function of α_1 and $\dot{\alpha}_1$, like in PD controllers :

$$\ddot{p} = k_p\alpha_1 + k_d\dot{\alpha}_1. \tag{2.34}$$

With this control signal, the linearized system described by equation 2.33 becomes :

$$\ddot{\alpha}_1 - k_d\dot{\alpha}_1 - (1 + k_p)\alpha_1 = 0. \tag{2.35}$$

The control will be efficient if $\alpha_1 = 0$ becomes a stable equilibrium of this equation. Standard linear stability analysis allows to state that the pendulum can be maintained in its unstable position if

$$k_d < 0,\ k_p < -1. \tag{2.36}$$

In a more general case, LQR provides an optimal strategy to evaluate the gain parameters k_p and k_d. It also allows to generalize the control by taking into account the dynamics not only of the pendulum but also of the cart, as it can impose an additional objective which would force the cart to be in a particular position on the track.

This control method is adapted for controlling systems near equilibria where the dynamics are considered as linear. The linearization of the system Eq. (2.26) around $\alpha = \pi, \dot{\alpha} = p = \dot{p} = 0$ yields :

$$\dot{\mathbf{X}} = \mathbf{A}\mathbf{X} + \mathbf{B}\mathbf{U},$$
$$\mathbf{X} = [\alpha - \pi, \dot{\alpha}, p, \dot{p}]^T, \quad \mathbf{U} = [0, 0, 0, \dot{p}_c]^T,$$
$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & -\nu & 0 & -\mu \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\mu \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu \end{bmatrix},$$

where we have assumed $\gamma_c = \gamma_d = 0$, for the sake of simplicity. By applying the procedure described in Section 2.1.2 with similar cost matrices $(\mathbf{Q}, \mathbf{R})$, we obtained $K =$

$[-127.5, -822.6, 2234.7, 437.7]$ for the control Eq. (2.10). After the swing-up with Lyapunov-based controller, we apply LQR control when the angle is near the pendulum's unstable equilibrium *i.e.* $|\alpha_{LQR} - \pi| = 0.1$ rad.

We now summarize this section dedicated to model-based control strategies. Far away from the unstable equilibrium, the Lyapunov theory based controller will push the dynamics of the pendulum towards the stable manifold of the unstable equilibrium. As the pendulum approaches the upside-down position, the linear controller is used to complete the control. These strategies have been successfully deployed on the real cart-pendulum system with the video of control [2]. Since these approaches are well-known in the literature, experimental results are not further detailed.

In this section, we have proposed simple techniques to stabilize an inverted pendulum : they are straightforward, but necessitate the knowledge of the model of the cart-pole. In the next section, we describe machine learning algorithms to perform the control within a model-free approach.

2. https ://youtu.be/BAzXTSYR5ug

2.3 Reinforcement Learning methods

2.3.1 Reinforcement Learning Paradigm

Reinforcement Learning (RL) is a control approach for discretized dynamical systems. RL exploits the framework of Markov Decision Process (MDP), which is a discrete-time stochastic control process. It is an extension of the Markov processes. At the discrete time step i, there are four components in MDP (S, A, P, R) : a set of states $s_i \in \mathcal{S}$ of the dynamical system and its surroundings, a set of available actions $a_i \in \mathcal{A}$ that the system can actuate, unknown dynamics that define transition probability among the states (dynamics model) $P \in [0, 1] : s_i \times a_i \rightarrow s_{i+1}$, and set of emitted rewards $r_i \in \mathbb{R}$ on each transition (s_i, a_i, s_{i+1}). MDP framework is based on the markov property, implying that states in the future are not dependent on the past given the present. Unlike traditional model-based methods in control theory described before, RL can solve Markov Decision Processes without the explicit knowledge of transition probabilities among the states.

We refer to the internal decision maker who uses an RL algorithm as an **agent**, and the whole physical system as the **environment**. At each time step i, the agent (algorithm) chooses an action a_i according to the assessed current state s_i. After the actuation, the environment provides a new state s_{i+1} and a reward r_{i+1} as seen in Fig. 2.6. Reward r_{i+1} is a learning signal for the agent of the quality of taken action a_i at state s_i. Then the agent-environment interaction cycle repeats. During the learning process, the agent evolves in an environment and tries to maximize its cumulative reward R_τ during certain time horizon $[0, T]$, which is referred as an episode or a trajectory τ :

$$\begin{aligned} \tau &= (s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T) \\ R_\tau &= \sum_{i=0}^{T-1} \gamma^i r\left(s_i, a_i\right), \end{aligned} \tag{2.37}$$

where $0 < \gamma < 1$ is the discount factor which measures the importance of the future unitary reward in computing the expected cumulative reward. If γ is close to 1, then the agent would focus on all the rewards during the time horizon T, which may increase the return variance among the trajectories and make convergence slower ; however, if γ is close to 0, then the agent will care only about the imminent reward and act sub-optimally with respect to long-term rewards. γ also helps for mathematical stability in the update equation for the infinite time horizon T control problems.

Agent interacts with an environment in an episodic manner to avoid being stuck in local minima and to diversify the trajectories. Imagine the agent navigating in the labyrinth for an exit. If the agent ends up at the wrong pathway of the labyrinth, it is easier to restart (**reset**) from some arbitrary position in the labyrinth. At the end of an episode the environment restarts at some initial state s_0 drawn from the starting state distribution ρ_0.

The choice of an action follows the policy $\pi(a|s)$ which is the probability of taking action a while being in state s. The objective in RL is to determine the best policy $\pi^*(a|s)$ for the agent, that maximizes the total expected cumulative reward R_τ.

2.3.2 Classical Reinforcement Learning

In this section, we present the classical methods of Reinforcement Learning created a couple of decades ago [Watkins, 1989, Williams, 1992, Sutton et al., 1999, Peters and Schaal, 2008]. Modern algorithms use Artificial Neural Networks (ANN) for different purposes, yet at their core, they use

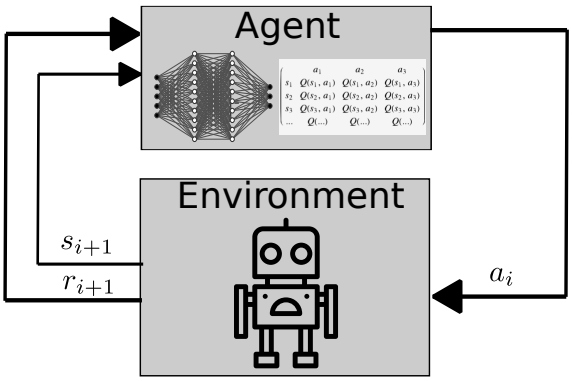

Figure 2.6 – RL interaction process. The state of the environment s_i is measured and given to the agent. The agent updates its policy and chooses accordingly the action a_i for the next step. Then the environment returns evolved state s_{i+1} and the reward r_{i+1} indicating if the new state is good or bad for the objective of an agent.

the principles defined in this section. The more detailed description of classical RL methods can be found in [Sutton and Barto, 2018].

2.3.2.1 Q-learning

In order to construct the policy $\pi(a|s)$, it is essential to estimate a reward-to-go [3] function R_i at step i, generally designed as the discounted cumulative reward :

$$R_i(\tau) = r_i + \gamma\, r_{i+1} + \gamma^2 r_{i+2} + \gamma^3 r_{i+3} + ..., \tag{2.38}$$

Since the cumulative reward depends on the states $s_i, s_{i+1}, s_{i+2}, ...$ and the actions $a_i, a_{i+1}, a_{i+2}, ...$, one can define an action-value function $Q(s_i, a_i)$ (Q refers to Quality) which computes the expected cumulative reward at the state s_i when performing the action a_i :

$$Q(s_i, a_i) = \mathbb{E}[R_i|(s_i, a_i)]. \tag{2.39}$$

This function could be the basis for constructing an optimal policy. For example a greedy policy will always select the best action $a^* = \underset{a\in\mathcal{A}}{\text{argmax}} Q(s_i, a)$ for an agent in the state s_i.

The learning process consists in visiting a large number of states and taking various actions, and to compute the reward expectation (2.39). However, it is usually time consuming and very difficult, if not impossible, to travel through all the states and actions to accurately determine the action-value function $Q(s, a)$, as it is necessary to sample the state and the action spaces to accumulate statistics for the rewards. In addition, a control task could be very long so it is not practical to wait till the end of the experiment and measure the cumulative reward, and update the function Q. In the MDP framework, one can rewrite Eq. 2.39 as [Sutton and Barto, 2018] :

$$Q(s_i, a_i) \approx r_i + \gamma\, Q(s_{i+1}, a_{i+1}). \tag{2.40}$$

Here we use the reward r_i after a sampled action a_i to represent the expected immediate reward, and $\gamma\, Q(s_{i+1}, a_{i+1})$ to represent the cumulative discounted future reward. In order to determine

3. sometimes referred as a return-to-go. It signifies the total cumulative reward until the end of the episode.

the action-value function, the agent interacts constantly with the environment during the learning phase and updates its Q function. This function can be updated through an iterative procedure :

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \Delta Q, \tag{2.41}$$

which is similar to the Euler scheme for numerically integrating the differential equation $\dot{Q} = \Delta Q$, where α plays the role of a time step. This is the idea of what is called the temporal difference learning (TD) approach [Sutton and Barto, 2018]. By defining $\Delta Q = Q^* - Q$, we know that the differential equation will drive Q to the target Q^*. The idea of the Q-Learning algorithm is to hypothesize that after sufficient amount of learning, the optimal action-value function is found :

$$Q^*(s_i, a_i) \approx r_i + \gamma \max_{a'} Q(s_{i+1}, a'), \tag{2.42}$$

with a' being the accessible actions at state s_{i+1}, which is consistent with the definition (2.40). It models that an approximation of the cumulative expected reward is the reward r_i plus the discounted cumulative reward at step $i+1$ by taking the best action $a^*_{i+1} = \text{argmax}_{a'}(Q(s_{i+1}, a'))$. To summarize, the Q-learning iterative procedure writes [Watkins, 1989, Watkins and Dayan, 1992] :

$$\underbrace{Q\left(s_i, a_i\right)}_{\text{new}} \leftarrow \underbrace{Q\left(s_i, a_i\right)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_i}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max Q\left(s_{i+1}, a\right)}_{\text{estimate of optimal future value}} - \underbrace{Q\left(s_i, a_i\right)}_{\text{old value}} \Big)}^{\text{temporal difference}}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}} \tag{2.43}$$

The parameter α measures the learning rate and the effect of the discount factor γ becomes even clearer : as it tends to zero, the learning agent only takes into account the immediate reward, while as γ is nonzero, the agent integrates future rewards in the learning phase. With this iterative approach, the agent learns while it evolves in the environment.

In order to avoid being stuck in a local optima, Q-learning employs an $\epsilon-$greedy policy during the learning process. The $\epsilon-$greedy policy is an interplay between *exploration* and *exploitation*. The agent exploits the policy or chooses the best action some of the time, and otherwise explores the consequences of randomly taken actions : at each time step, a random number $N_R \in [0, 1]$ is drawn, if $N_R < \epsilon < 1$, a random action is chosen ; otherwise the greedy policy is applied. It is a good practice to promote exploration in the early stage of the learning process with ϵ close to 1, with a small ϵ towards the end of the learning process. This means that it makes sense to explore more at the beginning of training when the agent does not have much information about how to maximize the reward-to-go, and to rely on the policy at the end while navigating the environment.

The learning happens according to the *generalized policy iteration* [Sutton and Barto, 2018], which alternates between **Policy Evaluation (PE)** or executing the policy in the environment and **Policy Improvement (PI)** Eq. (2.43). Generalized policy iteration is guaranteed to converge $Q \rightarrow Q^*$, given enough iterations (PE and PI). The schematic visualization of the procedure is in Fig. 2.7.

To store the expectation of the cumulative reward, the Q-Learning algorithm uses a Q-table that covers the whole state space and action space. This object takes the form of a huge matrix of dimension $N_s \times N_a$, where N_s is the number of discretized states, and N_a is the number of possible actions. This representation already underlines the limitation of this approach, because of the finite size of memory of modern computers.

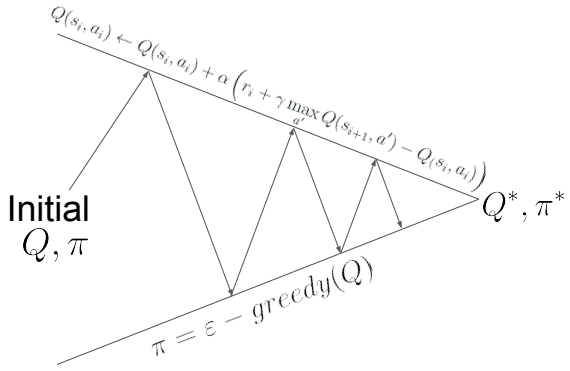

Figure 2.7 – Generalized policy iteration with action-value function Q that alternates between interaction with an environment and updating the action-value function Q

The whole algorithm is described in Algorithm 1. In the next section, the family of RL algorithms that operate directly on the policy π is investigated. These algorithms have their own advantages and disadvantages.

There exists variants of Q-learning such as Sarsa [see Eq. (2.45)] or Expected-Sarsa [see Eq. (2.44)] [Sutton and Barto, 2018], which yield somewhat similar results as Q-learning :

$$Q\left(s_i, a_i\right) \leftarrow Q\left(s_i, a_i\right) + \alpha\left(r_{i+1} + \gamma Q\left(s_{i+1}, a_{i+1}\right) - Q\left(s_i, a_i\right)\right) \tag{2.44}$$

$$Q\left(s_i, a_i\right) \leftarrow Q\left(s_i, a_i\right) + \alpha\left(r_{i+1} + \gamma \sum_a \pi\left(a \mid s_{i+1}\right) Q\left(s_{i+1}, a\right) - Q\left(s_i, a_i\right)\right), \tag{2.45}$$

where Expected-Sarsa Eq. (2.45) take the mean of Q-values over all actions and Sarsa needs its policy to choose the next action a_{i+1} before updating action-value function Eq. (2.44).

Algorithm 1 Q-Learning (Sarsamax)

Input : policy π, positive number of training steps num_steps, learning rate α, exploration function $\epsilon(i)$, discount factor γ
Objective : Optimal action-value function Q
Initialize Q arbitrarily (e.g., $Q(s, a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Observe s_0 after environmental reset
while Training **do**
    $\epsilon \leftarrow \epsilon(i)$
    Choose action a_i using ϵ-greedy policy derived from Q
    Take action a_i and observe r_{i+1}, s_{i+1}
    if s_i is terminal **then**
        $Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha(r_{i+1} - Q(s_i, a_i))$
    else
        $Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha(r_{i+1} + \gamma \max_a Q(s_{i+1}, a) - Q(s_i, a_i))$
    end if
end while

2.3.2.2 Policy Gradient Methods

Another strategy of the reinforcement learning is to operate directly on the policy $\pi_\theta(a|s)$ [Williams, 1992, Sutton et al., 1999, Sutton and Barto, 2018], which is a continuous differentiable distribution with parameters θ that maps input states $s \in S$ and actions $a \in A$ to the probability of execution. For the episode τ of T time steps, the objective of an agent is to find an optimal policy $\pi^*_\theta(a, s)$ that maximizes the total expected cumulative reward during an episode defined as :

$$\theta^\star = \arg\max_\theta \underbrace{\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_i \gamma^i r\left(s_i, a_i\right)\right]}_{R(\theta)}, \tag{2.46}$$

where $R(\theta)$ is an objective function to be optimized and $p_\theta(\tau)$ is a probability distribution of all the trajectories that incorporates the state transition probability of the MDP (environmental stochasticity) and the probability of a trajectory happening according to the policy $\pi_\theta(\tau)$. Because the state transition probability of the MDP (P) is not known and can not be used in the optimization process, we omit the environmental state transition probability from the expectation.

Suppose we have the state visitation sequence with the policy π_θ :

$$s_0 \overset{a\sim\pi_\theta(.|s_0)}{\longrightarrow} s_1 \overset{a\sim\pi_\theta(.|s_1)}{\longrightarrow} s_2 \overset{a\sim\pi_\theta(.|s_2)}{\longrightarrow} \dots \tag{2.47}$$

The probability of trajectory τ happening is :

$$p_\theta(\tau) = p(s_0, a_0, s_1, a_1 ... s_{T-1}, a_{T-1}) = p(s_0)\pi_\theta(a_0|s_0) \prod_{i=1}^{T-1} p\left(s_i \mid s_{i-1}, a_{i-1}\right) \pi_\theta\left(a_i \mid s_i\right) \tag{2.48}$$

and a total reward during an episode,

$$r(\tau) = r\left(s_0, a_0, \dots, s_{T-1}, a_{T-1}\right) = \sum_{i=0}^{T-1} \gamma^i r\left(s_i, a_i\right). \tag{2.49}$$

For the sake of simplicity, we will suppose that discount factor $\gamma = 1$ in further derivations. The objective of an agent is to maximize the total cumulative reward over all trajectories, so the return function is defined as :

$$R(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau) r(\tau) d\tau. \tag{2.50}$$

One of the methods to maximize the return is to use the gradient ascent method, which requires taking the gradient of an objective function $R(\theta)$ and updating the θ parameters respectively. Taking the gradient step in the positive direction of an objective function $R(\theta)$ with respect to θ i.e $\nabla_\theta R(\theta)$ enables us to update the policy parameters : $\theta \leftarrow \theta + \alpha\nabla_\theta R(\theta)$ and maximize the return. The parameter α is a learning rate of an update, similar to Eq. (2.43). The gradient is as follows :

$$\nabla_\theta R(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(\tau) r(\tau)\right], \tag{2.51}$$

where the second equality is due to the fact that $\frac{\nabla\pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla(\log \pi_\theta(\tau))$. The basic interest in making the logarithm appear in the equation is that instead of taking the gradient of the

product $\nabla(\pi_\theta(\tau)) = \nabla(\pi_\theta(a_0|s_0)\pi_\theta(a_1|s_1)\pi_\theta(a_2|s_2)...)$, we have the sum of the gradients : $\nabla(\log \pi_\theta(\tau)) = \nabla(\log \pi_\theta(a_0|s_0) + \log \pi_\theta(a_1|s_1) + \log \pi_\theta(a_2|s_2)...)$. Note that the gradient of a transition probability (P) with respect to θ amounts to zero, if we take it into account for the calculation of the gradient of an objective function in Eq. (2.51). Thus, knowing the $\log \pi_\theta(\tau)$ gradient is sufficient for finding the gradient of the return function, enabling the use of gradient ascent algorithms to maximize the return of an agent.

The expectation in the gradient of objective function Eq. (2.51) can be estimated using monte-carlo (meaning episodic, after agent completes the whole episode, we infer some useful information) sampling of N episodes ($j = 1...N$) :

$$\nabla_\theta R(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \left(\sum_{i=0}^{T-1} \nabla_\theta \log \pi_\theta \left(a_{j,i} \mid s_{j,i} \right) \right) \left(\sum_{t=0}^{T-1} r \left(s_{j,t}, a_{j,t} \right) \right). \tag{2.52}$$

The gradient ascent of the objective increases the likelihood of actions that led to large returns. The gradient of objective function Eq. (2.52) with respect to policy parameters θ is similar to the Maximum Likelihood Estimation framework [Conniffe, 1988] in machine learning.

The term $\sum_{t=0}^{T-1} r \left(s_{j,t}, a_{j,t} \right)$ can incorporate much variance especially if the trajectories are long. There are a number of techniques to reduce this variance. Firstly, because of causality argument in MDP stating that the present actions can only affect the future rewards, the term $\nabla_\theta \log \pi_\theta \left(a_{ji} \mid s_{ji} \right)$ does not affect the rewards emitted before step i, so we can truncate the sum of rewards and obtain "reward-to-go" :

$$\begin{aligned} \nabla_\theta R(\theta) &\approx \frac{1}{N} \sum_{j=1}^{N} \left(\sum_{i=0}^{T-1} \nabla_\theta \log \pi_\theta \left(a_{j,i} \mid s_{j,i} \right) \right) \left(\sum_{t=i}^{T-1} r \left(s_{j,t}, a_{j,t} \right) \right) \\ &= \frac{1}{N} \sum_{j=1}^{N} \left(\sum_{i=0}^{T-1} \nabla_\theta \log \pi_\theta \left(a_{j,i} \mid s_{j,i} \right) \right) \left(\hat{Q}^\pi (s_{j,i}, a_{j,i}) \right). \end{aligned} \tag{2.53}$$

Note that $\left(\hat{Q}^\pi (s_{j,i}, a_{j,i}) \right)$ is one episode monte-carlo approximation of the true action-value function under the current policy $Q^\pi (s_{j,i}, a_{j,i})$. To reduce the variance, it is a common practice to subtract from right-hand side of Eq. (2.53) a term called *baseline* [Greensmith et al., 2004] that is not dependent on θ, but that can affect the term $\sum_{t=i}^{T-1} r \left(\mathbf{s}_{j,t}, \mathbf{a}_{j,t} \right)$. This will permit to decrease the variance without changing the expected value of the gradient of the objective. The common choice for the baseline is the value function $V^\pi (s_t) = \mathbb{E}_{s_{t+1} \sim P, a \sim \pi_\theta} \left[\sum_{t=i}^{T-1} r \left(\mathbf{s}_{j,t}, \mathbf{a}_{j,t} \right) \right]$ corresponding to the total cumulative reward following the known greedy policy π_θ, with next states sampled from the dynamics (P) of MDP.

The mathematical derivation for why we can subtract a baseline such as the mean expected reward $\left(b = \frac{1}{N} \sum_{i=1}^{N} r(\tau) \right)$ is :

$$\mathbb{E} \left[\nabla_\theta \log p_\theta(\tau) b \right] = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) b d\tau = \int \nabla_\theta p_\theta(\tau) b d\tau = b \nabla_\theta \int p_\theta(\tau) d\tau = b \nabla_\theta 1 = 0$$

The difference between the reward-to-go of the current trajectory and the expected reward-to-go is called *Advantage* : $\hat{A}^{\pi_\theta} (s_i, a_i) = \delta_t^V = \sum_{t=i}^{T-1} r \left(s_t, a_t \right) - \frac{1}{N} \sum_{j=1}^{N} \sum_{t=i}^{T-1} r \left(s_{j,t}, a_{j,t} \right)$. It measures how good the current episodic return is with respect to the average return. It is a one-trajectory monte-carlo estimate of a true advantage function A^π. The gradient of Eq. (2.53) points

in the positive direction of good actions $\pi_\theta(a_i|s_i)$ if $\hat{A}^{\pi_\theta}(s_i, a_i) > 0$. The true Advantage function at time step i is estimated by the following :

$$A^\pi(s_i, a_i) = Q^\pi(s_i, a_i) - V^\pi(s_i) = r_{i+1} + \gamma V_t^\pi(s_{i+1}) - V_t^\pi(s_i). \tag{2.54}$$

This update is similar to TD update in Q-learning Eq. (2.43).

The basic form of policy gradient algorithms is described in the algorithm below.

Algorithm 2 Policy Gradient (REINFORCE)

Input : policy π parameterized by θ, learning rate α
Objective : Find optimal policy π^*
Initialize policy parameters θ
Observe initial state s_0
$i \leftarrow 0$
while Training **do**
    Generate one trajectory on policy using policy $\pi_\theta : s_1, a_1, r_2, s_2, a_2, \ldots, s_T$
    For $i = 1, 2, \ldots, T$ compute the advantage estimate $\hat{A}_i^\pi$ (e.g., $r_{i+1} + \gamma V_i^\pi(s_{i+1}) - V_i^\pi(s_i)$)
    Update policy parameters : $\theta \leftarrow \theta + \alpha \hat{A}^\pi \nabla_\theta \log \pi_\theta(a|s)$
end while

Compared to Q-learning or Value-based learning methods, the policy gradient family of RL algorithms has several advantages :

1. **Continuous actions** : Neural-network based function approximations permit to train these algorithms for MDPs with continuous actions unlike Q-learning algorithms that are not designed for continuous action environments.
2. **Performance in high-dimensional state spaces** : Policy gradient algorithms may perform better than traditional Q-learning based algorithms with environments in high-dimensional states, because they do not need to store the action-value estimation for all states. Basic version of the policy gradient algorithm does not need to store a Q-table for its operation. This significantly improves the memory imprint of an algorithm.
3. **Stochastic policies** : Policy gradient algorithms always have continuous and differentiable policies π_θ to ensure the existence of the gradient. Neural networks approximating π_θ incorporate stochasticity. Policy is designed as a Diagonal Gaussian in case of continuous actions and categorical (Bernoulli) distribution for discrete action MDPs. Policies as distributions may explore an environment better than randomly exploring with $\epsilon - greedy$ policy in Q-learning.

Policy gradient algorithms also incorporate various drawbacks :

1. **Instability** : As discussed before, the update Eq. (2.53) has a large variance among the sampled trajectories that necessitates more fine-tuning than Q-learning. If ANNs approximate the policy, the policy may diverge or be stuck in local point of optimization because of the gradient update.
2. **Hyper-parameter sensitivity** : Policy gradient algorithms based on ANNs require careful gradient updates and clipping the gradient values if they are large. These methods are more likely to diverge due to the bad hyper-parameters than the value-based methods.

Policy gradient algorithms are mainly used with neural networks representing the policy π discussed in the next section and in Section 2.3.3.4. A policy can also be parametrized by other function approximations such as gaussian processes [Ghavamzadeh et al., 2016]. Policy gradient methods are more difficult to implement than Q-learning methods and may imply instabilities that arise when the gradients of updates are large and the updated policies diverge.

In the next section, *Actor Critic* family of RL algorithms is discussed that incorporates the benefits of both Q-learning and policy gradient methods.

2.3.2.3 Actor Critic Methods

The main idea behind Actor Critic methods is approximating the Value function of an Advantage expression in Eq. (2.53) with a neural network [Peters and Schaal, 2008]. When a neural net is used to approximate the value function of an advantage function in Eq. (2.53), the Actor-Critic algorithm has a dual optimization problem of optimizing a policy and value functions. Dual optimization is subject to instabilities similar to GAN [Goodfellow et al., 2014] training. Actor is referred to the policy neural network that acts in the environment, and critic approximates the value function $V(s)$ i.e. the total cumulative reward following the best strategy from the current state.

It has been shown in [Schulman et al., 2016] that the advantage estimator reducing significantly the variance of Eq. (2.53) at time step t is the Generalized Advantage Estimation (GAE), which is the weighted sum of advantages from the current step till the end of the episode :

$$\begin{aligned}
\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} :=& (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \dots\right) \\
=& (1-\lambda)\left(\delta_t^V + \lambda\left(\delta_t^V + \gamma\delta_{t+1}^V\right) + \lambda^2\left(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V\right) + \dots\right) \\
=& (1-\lambda)\left(\delta_t^V\left(1 + \lambda + \lambda^2 + \dots\right) + \gamma\delta_{t+1}^V\left(\lambda + \lambda^2 + \lambda^3 + \dots\right)\right. \\
& \left. + \gamma^2\delta_{t+2}^V\left(\lambda^2 + \lambda^3 + \lambda^4 + \dots\right) + \dots\right) \\
=& (1-\lambda)\left(\delta_t^V\left(\frac{1}{1-\lambda}\right) + \gamma\delta_{t+1}^V\left(\frac{\lambda}{1-\lambda}\right) + \gamma^2\delta_{t+2}^V\left(\frac{\lambda^2}{1-\lambda}\right) + \dots\right) \\
=& \sum_{l=0}^{T} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned} \quad , \tag{2.55}$$

where $0 < \lambda < 1$; $\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V = -V(s_t) + r_t + \gamma r_{t+1} + \dots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k})$ is a k-step Advantage estimator and $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$ is a TD residual of V with discount γ. $\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)}$ is used in many actor-critic algorithms such as PPO [Schulman et al., 2015].

2.3.2.4 Partial summary

We have presented three families of RL algorithms : Q-learning, policy-gradient and Actor-Critic. Learning in tabular setting with Q-tables can be challenging due to the lack of state aggregation [Li and Zhou, 2013] or generalization. As seen in [Sutton and Barto, 2018] and Section 2.4.1, tabular Q-learning take many environmental steps to converge. This is due to the fact that Q-tables are updated one state at a time Eq. (2.43); while in practice, neighboring states may benefit from this update. One of the most effective ways to generalize among the neighboring states is to use neural networks to approximate the action-value function. In the next section, modern RL methods are presented that benefit from recent advances in deep learning.

2.3.3 Model-free deep reinforcement learning

In this section, we expand on the model-free deep Reinforcement Learning methods. These methods use several neural networks to approximate the action-value $Q(s,a)$, value $V(s)$ and policy $\pi(a|s)$ functions. The number of neural networks and their purpose varies from one RL

algorithm to another. Before diving into these algorithms, we remind the functioning of neural network.

2.3.3.1 Artificial Neural Networks

Artificial Neural Networks (ANN) is an assembly of idealized biological neurons [Sutton and Barto, 2018]. Dense Neural Networks (DNN) or Fully-Connected Neural Networks (FCN) are a specific type of ANN that has all of their neurons fully connected to each other. We refer hereby to the Multi-Layer Perceptron (MLP), the most common type of DNN where the neural net is divided into many nodes (neurons) and multiple layers Fig. 2.9. Each neuron in a layer is connected to every neuron in the previous and next layers. Each neuron, labeled k, possesses a state S_k and receives a signal p_k from other neurons. This incoming information writes :

$$p_k = \sum_j S_j w_{jk} + w_{0k},$$

where w_{jk} measures the weight of the link between the neurons j and k. In general, there is also a bias w_{0k} for each neuron. The incoming signal p_k is treated through a function f to define the new state of the neuron :

$$S_k \leftarrow f\left(p_k\right),$$

where f is the activation function which is almost always nonlinear. Then the process repeats until the output layer, each neuron in a consequent layer computes a linear matrix multiplication of its input (previous layer output) followed by an activation function.

The role of activation functions is to add non-linearity to the neural nets which helps to model complex and non-linear relations in the data. Some popular activation functions are ReLU [Hahnloser et al., 2000], Tanh (Tangent Hyperbolic), ELU [Clevert et al., 2016] and softmax [Goodfellow et al., 2016]. We plot some of them for visualization in Fig. 2.8.

The layers of neurons located in-between the input and the output layers are called *hidden layers*. For example in Fig. 2.9, the DNN has 5 inputs, 2 hidden layers and 3 outputs. Depending on the purpose of the output, activation function of the last layer changes. For example, if the DNN approximates the policy $\pi(a|s)$ that is bounded, then the last activation function may be sigmoid [Hornik et al., 1989] that is used for a last-layer classification models in deep learning. But if DNN serves as $Q(s, a)$ estimate, then the last layer activation function may be ReLU which outputs only positive quantity. The learning process happens via back-propagation [LeCun et al., 1998] of the error (loss) of the predicted value by the neural net and expected quantity via the gradient descent algorithms. The most popular algorithm of updating weights in a neural net is Adam [Kingma and Ba, 2014] which uses gradient descent update with momentum; other examples are AdaGrad [Duchi et al., 2011] and RmsProp [Duchi et al., 2011].

Loss function measures how accurate is the prediction of a neural net in comparison with the expected data. Typical loss functions for continous-output values from NNs are MSE (mean squared error), MAE (mean absolute error) or Huber Loss. Typical loss for probability output from NNs ($[0, 1]$) is a Cross-Entropy loss.

Let $Y = [y_1, y_2...y_N]$ be a *ground truth* and $\hat{Y} = [\hat{y}_1, \hat{y}_2...\hat{y}_N]$ be an output vector from a neural net, then MSE is defined as :

$$L(Y, \hat{Y}) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}_i\right)^2. \tag{2.56}$$

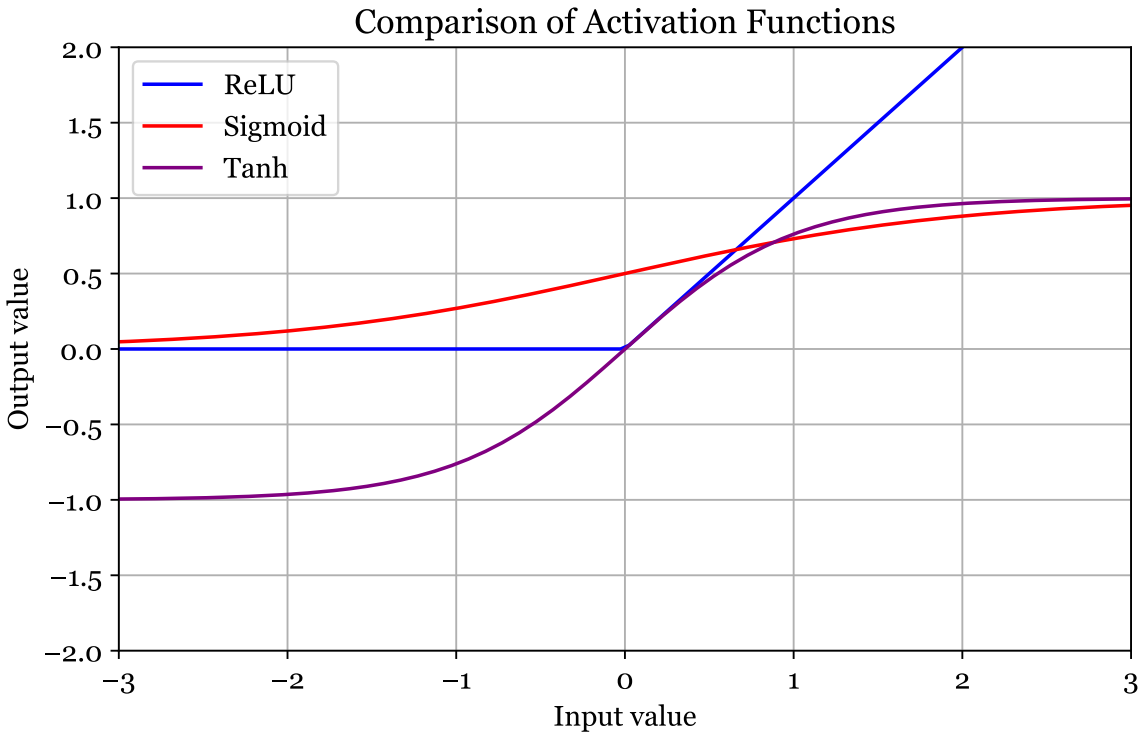

Figure 2.8 – Examples of activation functions in ANNs.

The loss function and a neural net can be differentiated and the direction of a minimum found. In simple cases, like a linear regression analytical solution can be found, but in neural nets with multiple inputs and complex nonlinearities, the analytical solutions become impractical. The idea behind back-propagation is that we calculate the gradient in the direction of a minimum of a loss function and slightly update the weights of a neural net in this direction. After this update, the prediction is expected to be slightly better.

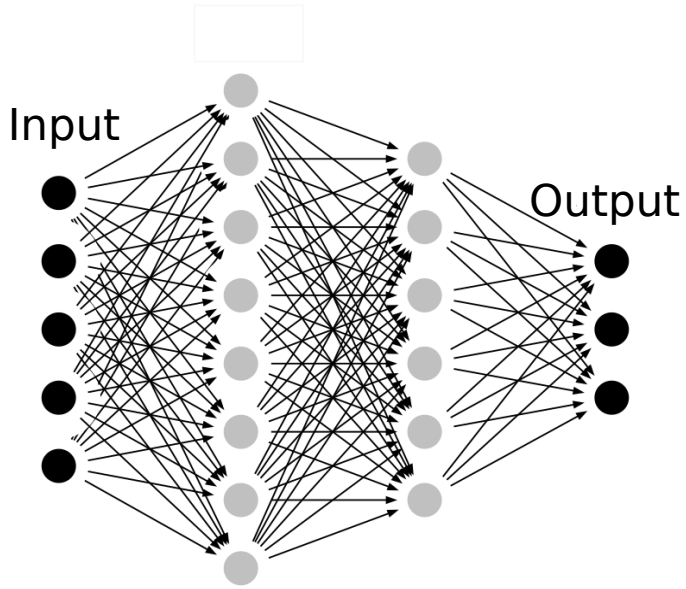

Figure 2.9 – Example of a neural network. It gets 5-dimensional vector as an input and outputs a 3-dimensional vector.

2.3.3.2 DQN

DQN [Mnih et al., 2013, Mnih et al., 2015] is a deep RL algorithm that uses a neural network to approximate an action-value function $Q(s, a)$. In addition, to stabilize the learning process and obtain more reliable results, DQN also employs a number of additional techniques such as replay buffer, fixed Q-targets and gradient clipping which improves the stability of learning by clipping the TD error Eq. (2.57) to [-1,1] interval.

The learning process happens at the end of every episode through the use of gradient descent (Adam) applied on mini-batches of transitions (s_i, a_i, r_i, s_{i+1}) sampled from a buffer of transitions. The replay buffer permits to reuse the previous transitions in an update, and also to break temporal correlation of these transitions for the learning process by shuffling the data.

DQN uses TD-learning with a neural network that approximates $Q(s, a)$. To avoid the target value (in TD update Eq. (2.57)) that changes frequently over time, the fixed network is introduced, thus decoupling the target value from the weight update. This increases robustness and stability of the learning. The weight updates of the local neural network follows the steepest gradient scheme :

$$\Delta\left(w_{jk}\right) = \alpha(\underbrace{r_i + \gamma \max_a Q\left(s_{i+1}, a_{i+1}, w_{jk}^{-}\right)}_{\text{Target}} - \underbrace{Q\left(s_i, a_i, w_{jk}\right)}_{\text{Local network value}})\nabla Q\left(s_i, a_i, w_{jk}\right), \tag{2.57}$$

where w_{jk} refers to the local network parameters, w_{jk}^{-} refers to the target network parameters. The TD target approximates the true $Q(s_i, a_i)$ in Fig. 2.11, and the update is done proportionally to the error between the approximated true action-value function and the current value. After C time steps (Table A.2), the target network parameters w_{jk}^{-} are updated with the local network parameters w_{jk} [Mnih et al., 2015]. The loss function for the backpropogation is the mean squared error between the target and the local network predictions.

To resume, the algorithm in fully-observed MDP is Algorithm 3 : In the next section, we

Algorithm 3 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights ω
Initialize target action-value function $\hat{Q}$ with weights $\omega^- = \omega$
for episode = 1 to M **do**
    Reset the environment s_0 and update exploration ratio ϵ
    for $t = 1$ to T **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(s_t, a; \omega)$
        Execute action a_t in the environment and observe reward r_t and state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
        Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \omega^-\right) & \text{for non-terminal } s_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(s_j, a_j; \omega))^2$ according to Eq. (2.57)
        Every C steps reset the target network with the local : $\hat{Q} = Q$
    end for
end for

describe a similar algorithm that improves incrementally on DQN.

2.3.3.3 DDQN

DDQN [Van Hasselt et al., 2015] is simple but effective way to improve upon the DQN method. Traditional DQN tends to overestimate the Q-values due to the *max* function of the update equation 2.43. DDQN uses the paradigm of *Double Q-learning* described in [Sutton and Barto, 2018]. Instead of taking directly the max Q value, we take the best action determined from the local Q-network, then compute the target using this action with the target Q-network. Since the two NNs are decoupled, this procedure avoids the bootstrap of a single NN for the TD update of Eq. (2.58).

When the training first begins, there's insufficient information to accurately determine the best actions to maximize the return. Therefore, taking the maximum of action-value function $Q(s, a)$ (which is noisy) as the best action to take, can lead to sub-optimal actions. DDQN on the contrary tackles the issue by modifying the target value in TD error. The TD error at time step i looks :

$$\Delta\left(w\right) = \alpha\left(\underbrace{r_i + \gamma Q\left(s_{i+1}, \operatorname{argmax} Q\left(s_{i+1}, a, \boldsymbol{\omega}\right), \boldsymbol{\omega}^-\right)}_{\text{Target}} - \underbrace{Q(s_i, a, \omega)}_{\text{Local network value}}\right) \nabla Q\left(s_i, a_i, w\right). \tag{2.58}$$

We remind the DQN update rule for comparison :

$$\Delta\left(w\right) = \alpha\left(\underbrace{r_i + \gamma \max_a Q\left(s_{i+1}, a_{i+1}, w^-\right)}_{\text{Target}} - \underbrace{Q\left(s_i, a_i, w\right)}_{\text{Local network value}}\right) \nabla Q\left(s_i, a_i, w\right).$$

In the next subsection, we describe the popular policy optimization algorithm PPO that directly optimizes the agent's policy. It is mainly used in an actor-critic form.

2.3.3.4 PPO

We describe the variant of proximal policy optimization (PPO), an Actor-Critic algorithm [Schulman et al., 2017, Ruckstiess et al., 2008] with the advantage normalization which is implemented in the open-source library [Raffin et al., 2019a]. The pseudo-code of the algorithm (PPO) is illustrated in the Algorithm 4.

Before describing the algorithm, we explain the concept of the entropy of the policy. Entropy at state s_t is defined as $H\left(\pi\left(\cdot \mid s_t\right)\right) = -\sum_{a \in A} \pi\left(a \mid s_t\right) \log \pi\left(a \mid s_t\right)$. High-entropy policy permits to better explore the environment and have more stochastic policy that would achieve the same return. High-entropy policies are more robust than deterministic ones, because if the system experiences perturbations, the high-entropy policy adapts faster. Moreover, because of their stochastic nature, high-entropy policies permit to better explore the environment compared to deterministic ones.

The algorithm follows actor-critic framework described in Section 2.3.2.3. On one hand, the actor neural network, denoted as actor NN, embodies the policy function ($\pi_\theta(s,a)$), which undergoes periodic updates through a refined policy gradient method that incorporates generalized advantage estimation $\hat{A}^{\mathrm{GAE}(\gamma,\lambda)}$ which became the standard variance reduction technique [Konda and Tsitsiklis, 2000] for stability and acceleration of training. This estimation serves as a metric to gauge $Q(s,a)$, the return of the chosen action in comparison to the expected return of the current state $V_\omega(s)$. To ensure a stable and robust training phase, PPO employs the so-called clipped surrogate objective function and incorporates the entropy bonus, enabling a balance between exploration and exploitation during the learning process. On the other hand, the critic NN operates as an evaluator, assessing continuously the value function used in the advantage expression of the update Eq. (2.54) of the actor network and providing feedback for policy optimization.

Finally, the loss being minimized to update the neural network parameters θ, ω is :

$$L_t(\theta) = L_{policy}(\theta) + c_v.L_{value}(\omega) + c_e.L_{entropy}, \tag{2.59}$$

where

$$L_{\mathrm{policy}}(\theta) = -\hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right], \tag{2.60}$$

$$L_{\mathrm{value}}(\omega) = \hat{\mathbb{E}}_t\left[\left(V_\omega(s_t) - \hat{R}_t\right)^2\right], \tag{2.61}$$

$$L_{\mathrm{entropy}} = \hat{\mathbb{E}}_t\left[\pi(a_t|s_t)\log\pi(a_t|s_t)\right]. \tag{2.62}$$

$L_{\mathrm{policy}}(\theta)$ is the policy surrogate loss, and PPO restricts the update to be within a small range defined by ϵ to prevent too large policy updates, which improves the stability in training. In Eq. (2.60), $r_t(\theta)$ is the probability ratio, defined as $\frac{\pi_\theta(s_t,a_t)}{\pi_{\mathrm{old}}(s_t,a_t)}$, representing the probability of taking action a_t at state s_t in the current policy compared to the previous one ; $\hat{A}_t$ is an advantage estimator and $\hat{R}_t$ is a return-to-go of the trajectory at time step t. $L_{\mathrm{value}}(\omega)$ is the value function loss, measuring the difference between the value function estimated by the current critic NN and the expected value $\hat{R}_t$ during the trajectories. Finally, incorporating the entropy bonus L_{entropy} encourages exploration by discouraging premature convergence to a deterministic policy. c_v and c_e are two coefficients to

Algorithm 4 Proximal Policy Optimization (PPO-Clip) Actor-Critic Style

Initialize policy parameters θ_0, rollout buffer $\mathcal{D}_{ep}$, initial value function parameters ω_0, clipping threshold ϵ.
while Training **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T time-steps and put data in $\mathcal{D}_{ep}$
    for epoch = 0, 1, 2, ... K **do**
        Take the entire episode of (s_t, a_t, r_t, s_{t+1}) transitions from $\mathcal{D}_{ep}$ and compute $\hat{A}_t$
        Optimize Loss function with respect to NN parameters θ , with T gradient steps using minibatch size M
        Update the value function by regression on MSE $\omega_{old} \leftarrow \omega$ using minibatches
        Update the policy $\theta_{old} \leftarrow \theta$
    end for
end while

weigh the value function loss and the entropy bonus, respectively. These are the hyperparameters that need to be tuned.

Note that this PPO actor-critic algorithm is on-policy, meaning that the algorithm does not reuse the experiences (s_t, a_t, r_t, s_{t+1}) from old policies. Unlike the experience replay buffer in off-policy algorithms such as DQN, PPO reuse only the experiences performed under the same policy to update the critic net, thus the policy optimization stability is guaranteed. PPO has the stability of trust region policy optimization algorithms [Schulman et al., 2015], being simpler and performing better. PPO is a very computation-efficient RL that has been used in a number of modern challenges such as playing DOTA2 [Berner et al., 2019] or fine-tuning LLM models such as ChatGPT [Ramponi, 2022].

PPO can be used for both discrete and continuous action environments because the policy π_θ is approximated with a neural network and all the loss functions work for both action types. Due to its simplicity and stability it can be a default deep RL algorithm for learning. In the next subsection, we explore the popular off-policy algorithm in model-free deep RL that is used for continuous-action environments.

2.2.3.5 SAC

Soft Actor Critic (SAC) [Haarnoja et al., 2017a, Haarnoja et al., 2019b, Haarnoja et al., 2019a] is an efficient actor critic algorithm that maximizes the return of a policy in the entropy-maximization framework [Haarnoja et al., 2017b]. With Actor Critic networks, it learns the optimal entropy of the policy while still maximizing the return. Its training is more stable and robust to hyperparameters than its popular predecessor DDPG [Lillicrap et al., 2016]. SAC has two $Q(s,a)$ networks mitigating the maximization bias of value-based learning (described in Section 2.3.3.3) and one policy network $\pi(a|s)$.

To apply the entropy maximization framework, the authors define soft value function :

$$V\left(\mathbf{s}_t\right) = \mathbb{E}_{\mathbf{a}_t \sim \pi}\left[Q\left(\mathbf{s}_t, \mathbf{a}_t\right) - \alpha \log \pi\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right], \tag{2.63}$$

which is an expected value function with the scaled (α) entropy maximization. The advantage of SAC is that it maximizes the *soft action-value function* [Haarnoja et al., 2017b] which maximizes the return making the policy as random as possible. Moreover, the learning mechanism for entropy hyperparameter α in Eq. (2.63) is proposed.

The algorithm optimizes 3 different losses :

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}}\left[\frac{1}{2}\left(Q_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right) - \left(r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p}\left[V_{\bar{\theta}}\left(\mathbf{s}_{t+1}\right)\right]\right)\right)^2\right], \tag{2.64}$$

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}}\left[\mathbb{E}_{\mathbf{a}_t \sim \pi_\phi}\left[\alpha \log\left(\pi_\phi\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right) - Q_\theta\left(\mathbf{s}_t, \mathbf{s}_t\right)\right]\right], \tag{2.65}$$

$$J(\alpha) = \mathbb{E}_{\mathbf{a}_t \sim \pi_t}\left[-\alpha \log \pi_t\left(\mathbf{a}_t \mid \mathbf{s}_t\right) - \alpha \overline{\mathcal{H}}\right]. \tag{2.66}$$

Equation (2.64) optimizes the two critics at the same time, Eq. (2.65) optimizes the policy following the entropy-maximization framework and Eq. (2.66) optimizes the entropy temperature. The overall algorithm is shown in Algorithm 5.

Algorithm 5 Soft Actor-Critic (SAC)

Initialize critic networks Q_{ω_1}, Q_{ω_2}, actor network π_θ, and target critic networks $Q_{\bar{\omega}_1}$, $Q_{\bar{\omega}_2}$
Initialize replay buffer $\mathcal{D}$
for each iteration **do**
    for each environment step **do**
        Select action $a_i \sim \pi_\theta(a|s_i)$ with exploration noise
        Execute a_i, observe next state s_{i+1}, reward r_i, and done signal d_i
        Store $(s_i, a_i, r_i, s_{i+1}, d_i)$ in $\mathcal{D}$
    end for
    for each gradient step **do**
        Sample mini-batch from $\mathcal{D}$
        Compute target value y using $Q_{\bar{\omega}_1}$, $Q_{\bar{\omega}_2}$
        Update Q_{ω_1}, Q_{ω_2} by minimizing loss i.e. $\omega_i \leftarrow \omega_i - \lambda_Q \hat{\nabla}_{\omega_i} J_Q\left(\omega_i\right)$
        Update π_θ by maximizing SAC objective
        Adjust entropy temperature $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$
        Soft update target networks $Q_{\bar{\omega}_1}$, $Q_{\bar{\omega}_2}$ i.e. $\bar{\omega}_i \leftarrow \tau \omega_i + (1-\tau)\bar{\omega}_i$
    end for
end for

2.3.3.6 Conclusion

We have presented the general theory for adaptation of classical Reinforcement learning methods with neural networks, rendering them model-free deep RL algorithms. We have presented a couple of algorithms such as DQN, DDQN, PPO and SAC. Other more recent model-free deep RL algorithms for discrete-action environments are Rainbow [Hessel et al., 2017], QR-DQN [Dabney et al., 2017] and for continuous-action environments is DROQ [Hiraoka et al., 2021]. In the Appendix B.2, we investigate other types of deep Reinforcement learning such as imitation learning and model-based deep RL. In the next section, we apply tabular and Neural-Network based Q-learning to successfully drive the real cart-pole system to its unstable equilibrium.

2.4 Controlling the pendulum using reinforcement learning

We aim at controlling an inverted pendulum in its unstable position, by model-free RL. The main advantage of model-free RL for control is that it avoids modeling the dynamics involved, unlike classical model-based approaches [Huang et al., 2012, Sun et al., 2018].

Many numerical studies have implemented an inverted pendulum virtual environment as a benchmark to test RL algorithms [Ope, , Koryakovskiy et al., 2017, Manrique Escobar et al., 2020, Zheng et al., 2020, Surriani et al., 2021, Özalp et al., 2020, Kumar, 2020, Baldi et al., 2020], but to our knowledge, there is no study that provides successful open-source RL implementations in experiments. First, except for a few studies that have discussed non ideal systems [Koryakovskiy et al., 2017, Manrique Escobar et al., 2020], most of these numerical implementations discard the effects associated to realistic (and thus more complex) control methods : in experiments, the control of the cart is subject to delay, hysteresis, biases and noise that can significantly alter the learning process. Second, most of the existing virtual environments consider only motion of the pendulum in a small angle range around the upward and unstable position and do not treat the whole control from the downward and stable position as expected in experiments. This ambition makes the control task significantly more difficult.

Although many Actor-Critic methods [Peters and Schaal, 2008] such as PPO [Schulman et al., 2017] could control successfully the cart-pole system, we focus on Q-learning and Deep Q-Network approaches due to their simplicity and easiness of understanding for educational purposes. We give insights about the implementation and the conditions of successful controls. Simulations with a virtual environment are provided to test the feasibility of the two approaches as well as to probe the effect of physical parameters that can not be easily tuned in experiments.

We provide all the material required to conduct the experiments detailed in this study, including an open-source code repository [Israilov et al., 2023] that enables the replication of all the approaches presented herein. It includes detailed instructions to build the prototype used in this work, configure its software interface and implement several controllers.

For the cart-pole problem, at each discrete time step $t_i = i\Delta t$ (where Δt denotes sampling time), the state s_i is given by the pendulum's orientation $\theta(t_i)$ and its angular velocity $\dot{\theta}(t_i)$, as well as the cart's position $x(t_i)$ and velocity $\dot{x}(t_i)$, *i.e* :

$$s_i = (\theta(t_i), \dot{\theta}(t_i), x(t_i), \dot{x}(t_i)). \tag{2.67}$$

According to the policy $\pi(a|s)$, the agent chooses and executes an action a_i which controls the cart movement for a given state s_i. This action changes the agent's state s_i to s_{i+1}, and the environment provides a reward r_{i+1} related to the proximity of the pendulum to its unstable position. This process is then iterated at stage $i + 1$: the loop is depicted in Fig. 2.10.

As discussed in Section 2.3.2.1, one of the intuitive ways for agent's learning is through the action-value function $Q(s, a)$. We discuss the limitations of the basic Q-Learning for this system, then the more advanced DQN approach is exposed and we show that it successfully maintains the pendulum at the target position in both experiments and simulations. Finally, we explore the influence of different system's physical parameters on the control quality in the virtual environment.

RL Environment

Our objective is to maintain the pendulum at the target position $\theta = \pi$ while centering the cart ($x = 0$) at the same time. The system state s has been defined with Eq. (2.67). To avoid an angle

discontinuity at $\theta \in \{-\pi, \pi\}$, $\sin(\theta)$ and $\cos(\theta)$ are given to the learning agent instead of only θ. Moreover, passing through $\sin(\theta)$ and $\cos(\theta)$ ensures that the inputs to ANN are in the standard range of $[-1, 1]$ of ANN. The inverted pendulum system is driven by a motor on the cart and it has direct control on the mean cart's velocity $\dot{x}$ via an applied voltage on the motor. Three actions are offered to the agent at each time step, *i.e*, $a_i = \{-U, 0, +U\}$, with $U \in [0, 12\text{V}]$ a fixed voltage. At each time step, the cart can translate in both directions or keep its current position, according to its dynamics.

We now proceed with the reward function. The reward is maximum as the objective is reached, *i.e*, the pendulum in its unstable position ($\theta = \pi$). In addition, we add the requirement for the cart to be centered around the middle of the track ($x = 0$). For this purpose, there are many options to design the reward function [Sutton and Barto, 2018], and for simplicity, we have chosen :

$$r(\theta, x) = (1/2)\,(1 - \cos(\theta)) - (x/x_0)^2\,, \tag{2.68}$$

where $x_0 < x_{max}$. This mechanical constraint of x_{max} does not prevent the agent to reach the control objective on the angle. The maximum of this function is equal to one, as $\theta = \pi$ and $x = 0$.

The normalized return of an episode is computed as the cumulative reward of the entire episode divided by the maximum episode length, *i.e.* 800 (time steps). Such a definition gives an evaluation of the policy : the closer to 1 the normalized return, the better the episode. An episode is interrupted when the state s_i meets at least one of the following conditions :

1. the dimensionless cart's position exceeds the physical boundaries, *i.e*, $|x| > x_{max}$; In this case, the agent is strongly penalized and the cumulative reward of the episode is reduced by -400.
2. the angular speed exceed 14 rad/s, since in practice, we would like to avoid the pendulum spinning too rapidly. This value has been chosen according to the mechanical limit of our experimental system.
3. the maximum duration $T_{ep} = 800\Delta t$ is attained, where $\Delta t = 0.05\,\text{s}$. This choice has been set to diversify the experience and avoid being stuck in local minimums, which corresponds to roughly 2 or 3 times of optimal swing-up time. These values are indeed adapted for an acceptable control quality. In the real experiment, one episode takes approximately $40\,\text{s}$.

At the beginning of every episode, we initialize the system with the cart and the pendulum at rest, *i.e*, $\theta = 0$ and $x = 0$. We ensure that the pendulum is at rest to learn proper swing-up strategies. Between two episodes, the system waits $120\,\text{s}$ to ensure that the condition $\theta = \dot{\theta} = 0$ is satisfied.

The learning process consists in accumulating statistics during successive episodes. Plotting the normalized return as a function of the episode number can be noisy and we smooth the data by performing a moving average in Q-learning and DQN over 300 and 30 episodes, respectively, as the former is less stable [see the raw "learning curve" figures in Fig. A.2].

Finally, we prefer to represent the learning curve by plotting the normalized return as a function of the total number of time steps to give insights about the true time of the learning process, because some episodes might not run to the end.

For simplicity, we deliberately choose to control the pendulum with discrete actions. We have tested two model-free RL algorithms, Q-learning [Watkins, 1989, Watkins and Dayan, 1992] and Deep Q-Network (DQN) [Mnih et al., 2013]. Both approaches are detailed in Section 2.3.3.2.

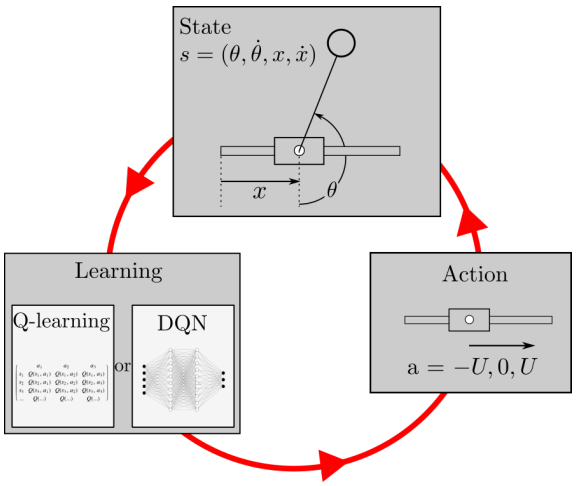

Figure 2.10 – RL learning process. The state of environment s is measured and given to the agent. The agent updates its policy and accordingly chooses the action a for the next step among $-U$, 0 or U. After the sampling time, the state s evolves and the cycle continues. In our case, $s = (\theta, \dot{\theta}, x, \dot{x})$. The policy is updated by modifying the action-value function for Q-Learning and DQN, using a so-called Q-table or dense neural networks, respectively.

2.4.1 Simulations results

In the experimental setup, the state information is gathered directly from the physical world, and the agent interacts with the environment via the Low-level Interface (LLI) [see Appendix A.1]. In the virtual setup, the agent's state is updated through eqs. 2.21, 2.22, 2.23. The effects of the voltage U, the dry friction f_c acting on the motorized base and the viscous friction k_v of the pendulum were investigated systematically in simulation.

In the experimental setup, the measurements are subject to the white noise and we also investigated the effect of the artificially added white noise upon the simulated angular state values. We have introduced a Gaussian noise to the measurement of the pendulum angle θ, *i.e*, at each instant $t = i\Delta t$, $\theta_i \sim \mathcal{N}(\theta_i^m,\, \sigma_\theta^2)$, where $\mathcal{N}$ refers to the normal distribution, $\theta_i^m = \theta_{i-1} + \dot{\theta}\Delta t$ is updated from the previous state. Naturally, a noise of amplitude $\sigma_{\dot{\theta}} = \sigma_\theta/\Delta t$ was then introduced to the $\dot{\theta}$ measurement. The influence of the noise amplitude σ_θ and $\sigma_{\dot{\theta}}$ on the control quality was then analyzed.

Q-learning

In Q-learning, the observation space $(\sin(\theta), \cos(\theta), \dot{\theta}, x, \dot{x})$ is discretized into different number of bins, whose sizes is matter of compromise. A Q-table with low resolution results in relatively fast simulations and limits the use of computer memory. On the other hand, the resolution needs to be high enough to ensure the success of the learning process. As an example we start with a sparse and homogeneous discretization with nBins = (10, 10, 10, 10, 10). In this case we expect the Q-table to contain $3 \cdot 10^5$ elements, given that there are three possible actions.

The Q-table size gives a minimal estimate of the total number of time steps to learn assuming that the agent needs to visit each element of the table. This number is 10-100 times higher in

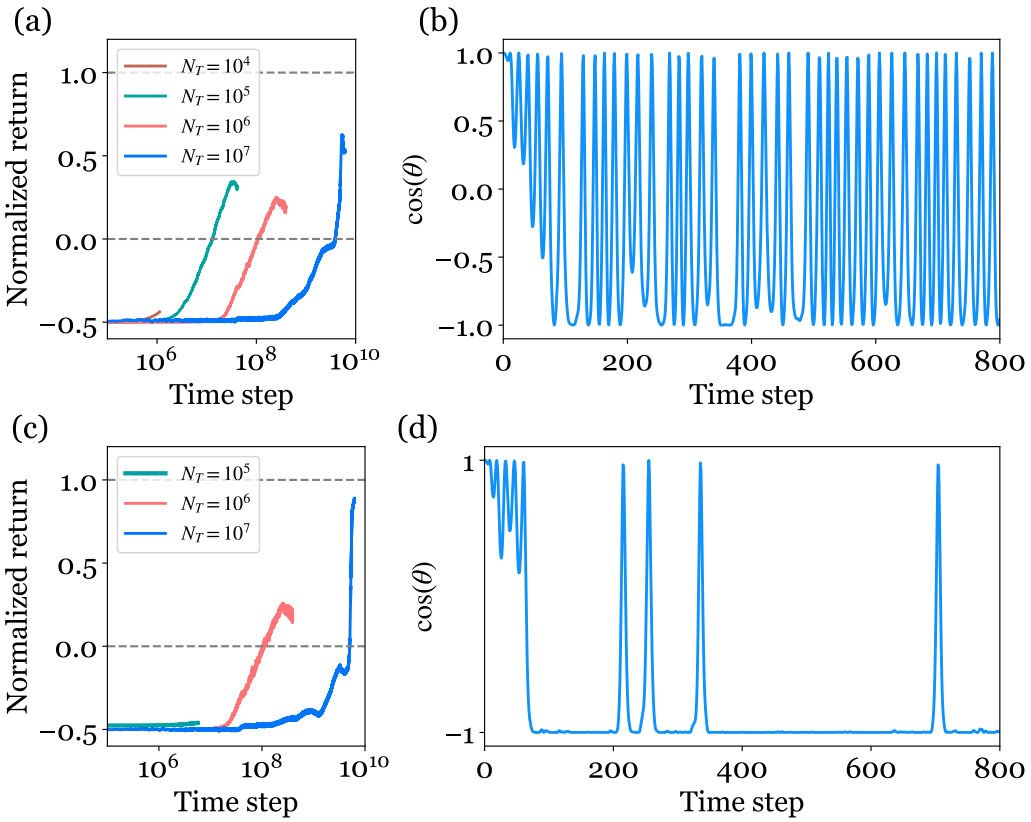

Figure 2.11 – Learning results using the basic tabular Q-learning implementation. Left : Normalized return as a function of the number of time steps for different total number of episodes N_T. Right : Temporal evolution of $\cos\theta$ in the best episode of the longest learning process ($N_T = 10^7$). The observation space $(\sin(\theta), \cos(\theta), \dot{\theta}, x, \dot{x})$ is discretized homogeneously into different number of bins : a) and b) nBins = (10, 10, 10, 10, 10), c) and d) nBins = (50, 50, 50, 10, 10).

practice given that the basic Q-learning algorithm usually suffers a low sample-efficiency [Sutton and Barto, 2018] : some elements are never evaluated while some others can be updated regularly.

We have tested the Q-learning approach in simulation with different total number of episodes N_T from 10^4 to 10^7. We recall that one episode contains 800 time steps at maximum ; the average number of time steps per episode is lower in practice due to numerous interrupted episodes at the beginning of the learning process. The technical details such as the value of the hyperparameters are found in Appendix A and Table A.2. Given the expression of the reward function and of the penalty, it can been inferred from Fig. 2.11 the cumulative reward spans from -0.5 (the cart goes quickly out of the track) to 1 (successful learning). This is related to the fact that at the beginning of the training, the agent almost always goes out of the track and receives a huge negative reward -400 ; given the normalization of 800 steps, we obtain -0.5 for the normalized return [4]. Below 10^6 time steps, the normalized return remains close to its minimum. The system requires at least 10^7 time steps (10^5 episodes) to observe an increase of the normalized return above 0 (Fig. 2.11a). Even in this case, the cumulative reward remains low, around 0.3, and reaches 0.55 at most as the number of time steps is increased to 10^{10}. For such an episode, the pendulum can be maintained at

4. Reminder : normalized return - cumulative reward of the entire episode divided by the maximum episode length

its vertical position only in a short amount of time, otherwise the pendulum oscillates (Fig. 2.11b). Transposed to experiments with a physical time interval $\Delta t = 0.05$s, 10^7 time steps correspond to 6 days of experiments ! We can nevertheless discuss the effect of the discretization of the Q-table, which is too low in the former example to reach a cumulative reward close to 1 even after a very large number of time steps. In the following, we estimate the typical value n_θ for the bin in θ : the discretization interval is $\Delta\theta = 2\pi/n_\theta$. In order to ensure the learning objective, the time interval separating two actions must not be too large with respect to this discretization. We expect that Δt should be smaller than the typical time the agent lasts in one interval : we can assess this duration in the limit of small damping. By assuming that the pendulum is weekly damped, we approximate the Eq. (2.21) with $\ddot{\theta} + \omega^2 \sin\theta = 0$. Consequently we write the energy conservation $\frac{1}{2}\dot{\theta}^2(t) = \omega^2 \left(\cos\theta(t) - \cos\theta(0)\right)$, where $\dot{\theta}(0) = 0$. Between two iterations the angle varies within an increment $\Delta\theta$ and we write $\theta(t) = \theta(0) + \Delta\theta$, $\Delta\theta \ll 1$:

$$\cos\theta(t) = \cos(\theta(0)) - \Delta\theta \sin\theta(0) + o(\Delta\theta)^2 \tag{2.69}$$

$$\dot{\theta} = \frac{\Delta\theta}{\Delta t}, \tag{2.70}$$

such that we deduce that :

$$n_\theta = \frac{\pi}{\omega^2 \Delta t^2} \frac{1}{\sin\theta(0)}. \tag{2.71}$$

This gives the order of magnitude $n_\theta \sim 50$. The presence of a divergence near the unstable equilibrium shows that the discretization must be refined at least near $\cos(\theta) = -1$.

Consequently, we tested a finer resolution nBins = (50, 50, 50, 10, 10) with $\sin(\theta)$, $\cos(\theta)$ and $\dot{\theta}$ discretized into 50 bins. The computation memory increases exponentially with the size of the Q-table and any finer resolution would be unpractical. As observed in Fig. 2.11c, it takes at least 10^8 time steps to see a normalized return above 0. After about 5.6×10^9 time steps (7×10^6 episodes), the system has finally learned reasonably well and obtain a normalized return of ~ 0.8 : the pendulum can stay in the goal position for a finite period, but quickly falls over to be quickly swung back up again (Fig. 2.11d).

The inefficiency of the learning is rationalized by the fact that the matrix representation of Q-Table is not adapted to solve the swing-up problem. To update the action-value function more efficiently, a better function approximation with state aggregation is needed. In that regard, artificial neural networks show very promising capabilities and is data efficient [Riedmiller, 2005].

To overcome this obstacle, it appears necessary to exploit a more efficient function approximator. Deep Q-Network (DQN) [Mnih et al., 2013] is a reinforcement learning algorithm based on the Q-learning approach that takes advantage of neural networks in place of the matrix "Q-table" to approximate the true "action-value" function. Neural networks provide an effective way to approximate $Q(s, a)$, because they can incorporate non-linearity and aggregate among the states due to the interconnection between the neighboring layers of the neural net. This leads to a more efficient action-value approximations. The algorithm is described in Section 2.3.3.2 and we remind the functioning of DQN for the cart-pole environment in Fig. 2.12.

2.4.1.1 Deep Q Learning

We implement the Deep Q-Learning technique. In this approach, the Q-Table for approximating the Q-function in Section 2.3.2.1 is replaced by an ANN, which is named Deep Q Network

(DQN). Similar to any other deep learning algorithm, the training of DQN depends on the hyper-parameters, which determine the network policy structure, the learning strategy and the learning speed. We offer a set of fixed hyper-parameters (table A.2), which is robust for our system.

All the simulations and experiments were driven by a Dell Precision 7550 using its internal GPU. For one simulation with 1.5 10^6 time steps with logging and evaluation loops, it takes 10.7 minutes using GPU (NVidia Quadro T2000), and 13.43 minutes using CPU only (Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz).

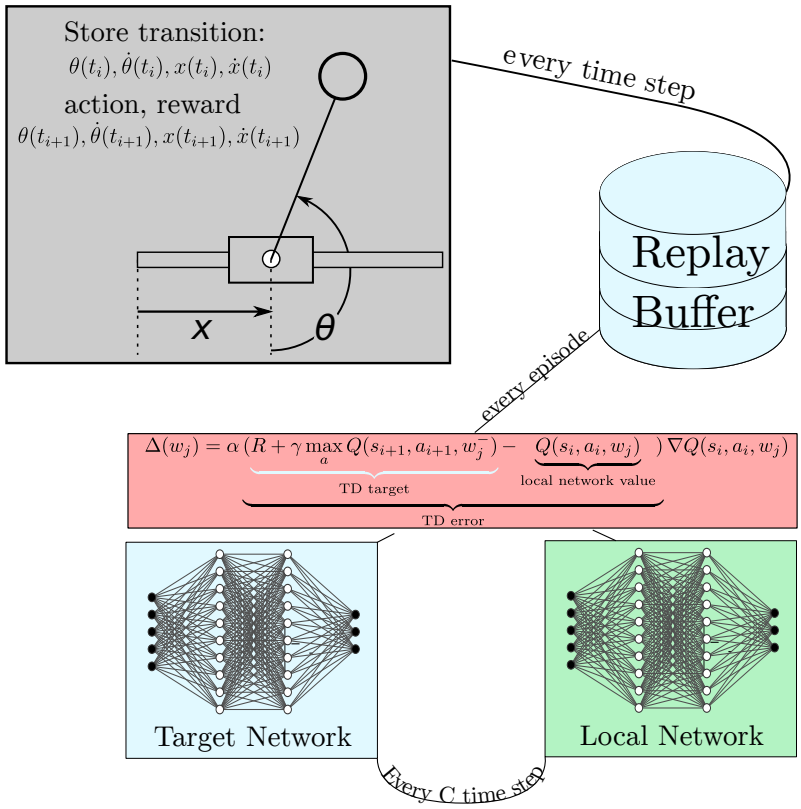

Figure 2.12 – DQN learning process. The state of the environment $s_i = (\theta, \dot{\theta}, x, \dot{x})$ is assessed and the policy (ANN) outputs the corresponding action a_i. Following this, the system is actuated, leading to reward r_i the observation of a new state s_{i+1}. The transition data comprising (s_i, a_i, r_i, s_{i+1}) is stored in the replay buffer. At the end of every episode, the agent updates its action-value function $Q(s, a)$ *i.e.* local network using gradient descent. Learning continues for C time steps, when the target network is replaced with more trained local network.

In our problem, the ANN's input layer have 5 neurons that handle the five components of the observation $(\sin(\theta), \cos(\theta), \dot{\theta}, x, \dot{x})$. Each of these five neurons is connected to the first hidden layer consisting of 256 nodes, which are also connected to a second hidden layer of also 256 nodes. For the two hidden layers, we use the Rectified Linear Unit function (ReLU) [Sutton and Barto, 2018, Mnih et al., 2015].

The network's output layer is made up of 3 neurons, which gather information from the previous hidden layer. Each output neuron represents the action-value of 3 possible actions for the

current state, visualized in Fig. 2.9. The training process updates the unknown parameters w_{jk} and w_{0k}, in order to minimize the error between the output of the ANN, and the estimated true value based on the real reward given by the environment.

In parallel to experiments, we performed simulations of the model. For both approaches, the features and quality of the learning process are evaluated. Note that the maximal number of time steps (150000) for the complete training is chosen so that the steady state average value is reached in both real and virtual experiments. We evaluate the policy performance every 5000 time steps with an inference. It consists in testing a greedy policy during one complete episode, with the initial condition $(\theta, \dot{\theta}, x, \dot{x}) = (0, 0, 0, 0)$. This protocol is applied directly in experiments, while in simulations, the inference curve consists in computing the evolution of the average normalized return of 10 episodes (instead of only one in experiments) with equidistant initial conditions : ($\theta_0 \in (-10°, 10°)$ and $(\dot{\theta}, x, \dot{x}) = (0, 0, 0)$). This allows to test the robustness of the policy in simulation, *i.e*, the capacity to generalize and achieve a high normalized return from different initial states, other than the particular initial state of the learning process. This protocol however is not viable in experiments since in practice it is difficult to control precisely the initial angle of the pendulum other than its equilibrium position. Finally, the best learned policy in the sequel corresponds to the DQN model that obtained the highest normalized return among all the inferences.

2.4.2 Experimental results

We first discuss the results of the outlined DQN algorithm obtained with the experimental setup. The only control parameter is the applied voltage U, which is directly proportional to the target cart's velocity value $\dot{x}_c$ in Eq. (2.23). Fig. 2.13 displays the temporal evolutions : (a) of the cart's position and (b) of the pendulum's angle during a single episode for the best learned policies. Two distinct voltages were tested : $U = 2.4\,\mathrm{V}$ and $U = 7.1\,\mathrm{V}$.

The voltage $U = 2.4\,\mathrm{V}$ is not sufficient to swing up the pendulum, and the best policy yields an oscillation of the pendulum around 0. This means that the energy provided with this voltage is not high enough to swing up the pendulum or that the total duration of one episode, 800 time steps, is not large enough to increase the maximal angle, period after period. Given that the maximum angle is reached after 300 time steps already, the first assumption is probably the good one.

For the other voltage $U = 7.1\,\mathrm{V}$, the cart initially oscillates with a large amplitude and the pendulum swings up after about the equivalent of almost 3 periods. As soon as the unstable equilibrium is reached, the cart turns into a vibration regime with smaller amplitude to maintain the pendulum balanced upward around $\theta = \pi$. The learning and the inference curves (see Fig. 2.14, thick solid lines) reveal exactly the same results that for $U = 7.1\,\mathrm{V}$, the normalized return in both learning and inference reaches a high plateau value of $\sim 0.8 - 0.9$, indicating a successful control, while for $U = 2.4\,\mathrm{V}$, the normalized return stays very low around 0.1.

Simulation results analysis

In this section, we perform simulations and test different important physical parameters which could influence the control quality. All the parameters are kept constant and defined with the Tables A.1 and A.2 except the one investigated. The voltage is set to $U = 12\,\mathrm{V}$ and the noise $\sigma_\theta = \sigma_{\dot{\theta}} = 0$ if not specified.

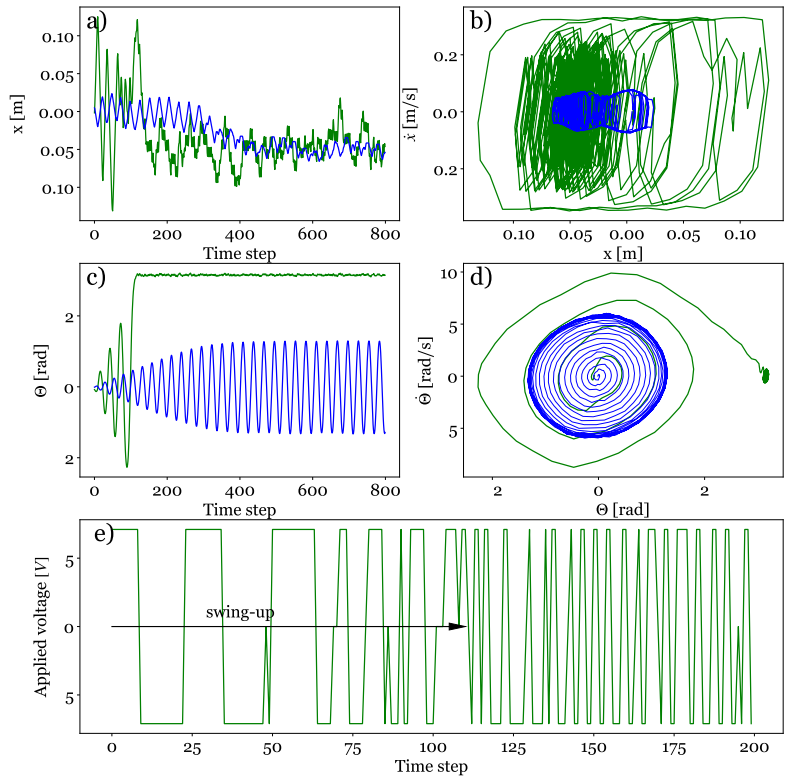

Figure 2.13 – Experimental results with the best policies in inference for two different applied voltages $U = 2.4$ V (blue) and 7.1 V (green) : a) Temporal evolution of the cart's position x during one episode. b) trajectory of the cart in the $(x, \dot{x})$ space. c) Temporal evolution of the pendulum's angle θ during one episode. d) Trajectory of the pendulum in the $(\theta, \dot{\theta})$ space. e) Temporal evolution of the applied voltage during the first 200 time steps.

Effect of action amplitude – applied voltage on the DC motor. We have shown with our experimental setup that the action amplitude plays a crucial role in the task : a low voltage applied on the DC motor results in a failure of control. Here we test a range of U from 2.4 to 12 V in the virtual environment and the results are presented in Fig. 2.14. First, we note that the simulation results are consistent with those found in experiments (thick curves), *i.e*, both normal and thick curves of ($U = 7.1$ V) as well as ($U = 2.4$ V) show similar trend. Fig. 2.14a displays the learning curves. The normalized return increases and then reaches a plateau for all the applied voltages. However, up to $U = 4.7$ V, the plateau value is smaller than 0.4, close to that observed using Q-learning algorithm, referring to an oscillation around the stable position. Above 4.7 V, DQN algorithm gives satisfying performance during the learning process.

To assess the performance of the optimal policy obtained for each applied voltage, we plot the inference results in Fig. 2.14b. Because there is no exploration and the optimal action is chosen at

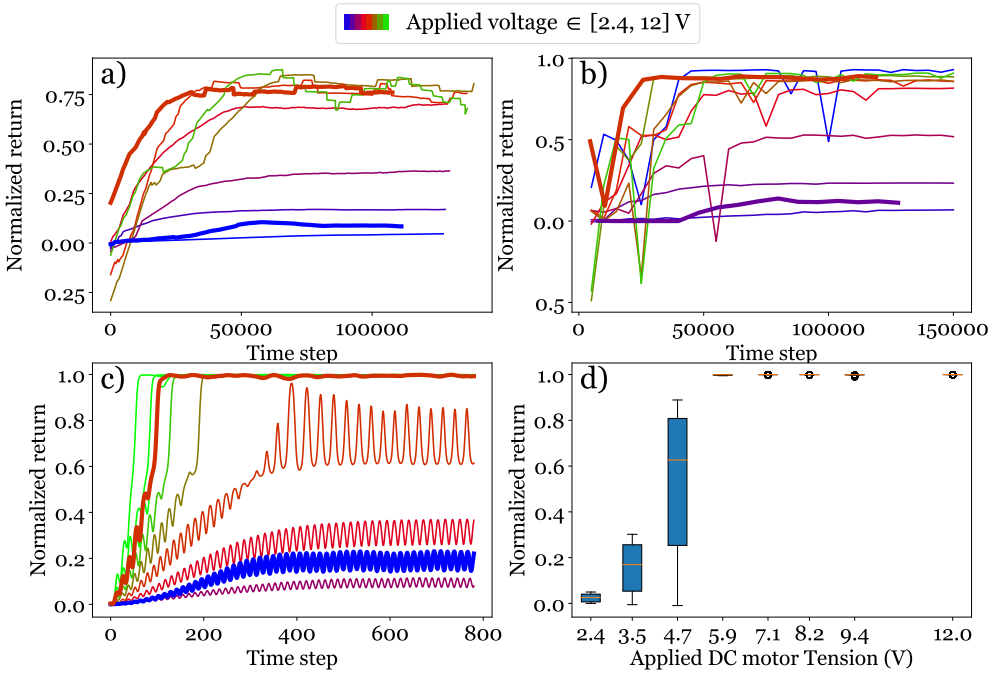

Figure 2.14 – Influence of the applied voltage on the learning process. Thin curves correspond to different simulations, while thick curves refer to the experimental observations. a) Learning curve. b) Inference curve built from inferences performed every 5000 time steps. c) Temporal evolution of the reward for the episode initiated at $\theta_0 = 0$ following the best learned policy. d) Statistics over 10 episodes initiated with θ_0 between $-10°$ and $10°$ of the plateau reward following the best learned policy.

each time step, the plateau value of each inference curve is expected to be greater than the corresponding learning curve. Nevertheless, some inferences exhibit negative peaks associated to the fact that within the set of 10 episodes, averaged to measure the normalized return of an inference, some of them are terminated by the cart reaching x_{max} and are strongly penalized consequently. These negative peaks disappear as the number of time steps increases and the learning process continues. A normalized return between 0.8-0.9 is a good value as it is calculated from the individual reward averaging on one episode, and this includes the initial stage before swing up. This can be seen in Fig. 2.14c where the learning process is probed by plotting the time evolution of the reward for an episode initiated at $\theta_0 = 0$ following the best learned policy obtained after the 150000 time steps. From $U = 5.9\,\mathrm{V}$, the plateau of the reward is around 1 and the system reaches the objective. This figure also reveals that the higher the applied voltage, the quicker the swing-up is.

To probe the robustness of the best learned policy for each applied voltage, we have measured the average of the plateau reward for 10 episodes initiated with equidistant initial values of θ_0 between $-10°$ and $10°$. Statistics over these 10 episodes are represented by a box-plot of the reward as a function of U (Fig. 2.14d). It shows that the pendulum can operate and maintain a swing up for some values of θ_0 even for $U = 4.7$V, but that this behavior becomes robust only for $U \geq 5.9$V.

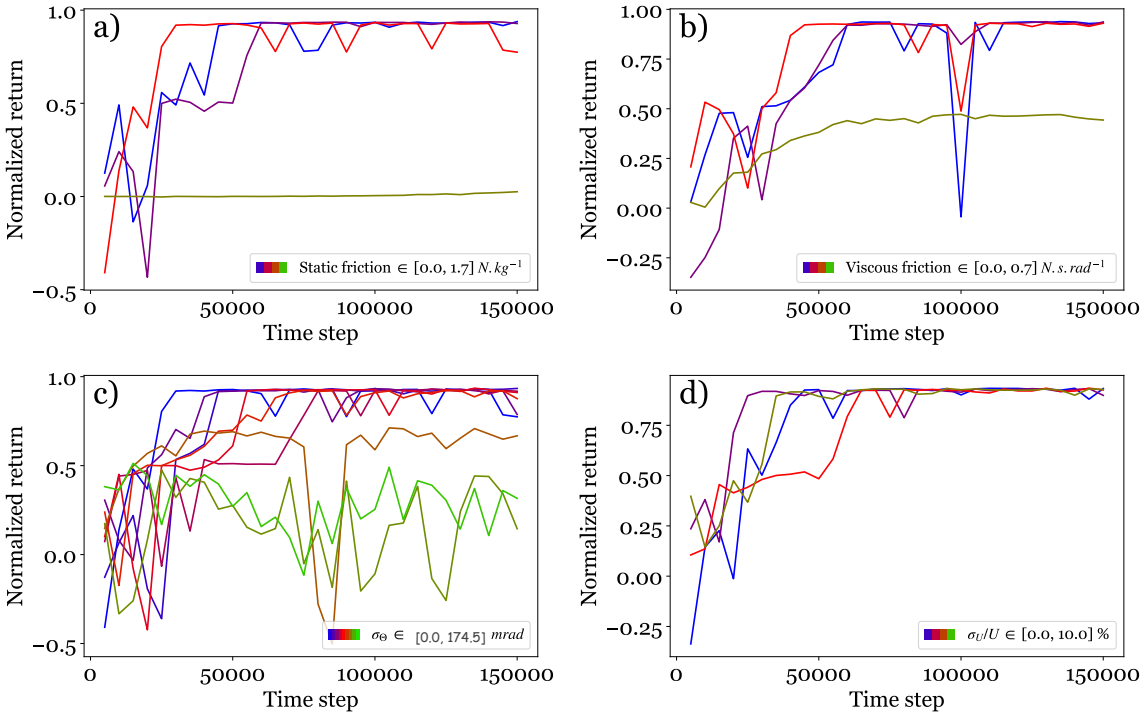

Figure 2.15 – Influence of the physical parameters on the control : inference curves of a) static friction, b) viscous friction, c) measurement noise and d) action noise.

Effect of the physical parameters In what follows, we numerically investigate the robustness of the learning process with respect to the two friction coefficients and to the two sources of noise.

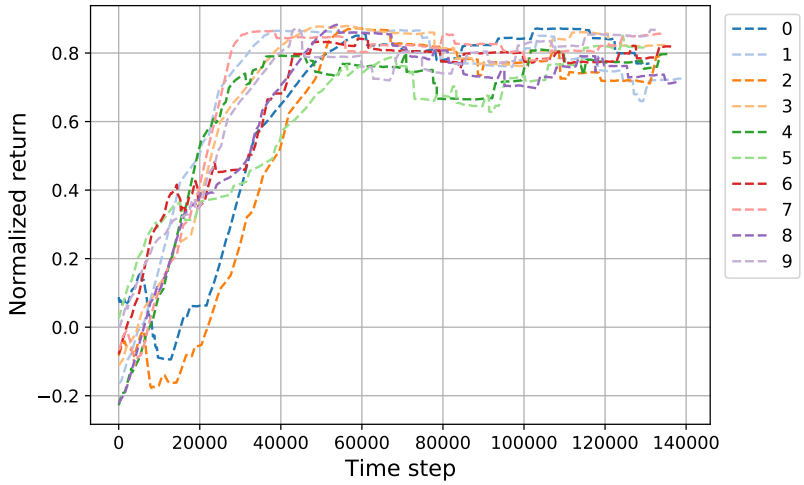

Figure 2.16 – Influence of the seed number of a random generator on the learning process. Results are shown for the DQN algoritm with the default parameters (Table A.2 and Table A.2), $U = 12$ V and $\sigma_\theta = \sigma_{\dot{\theta}} = 0$.

In Fig. 2.15a, the static friction is varied from 0 to 11.7 N.kg^{-1}, keeping the other parameters constant. We observe that the value 1.17 N.kg^{-1} measured with the real system does not perturb the learning process in comparison to a system without friction. However, increasing tenfold this parameter value prevents the system from learning correctly. In Fig. 2.15b, the viscous friction is varied from 0 to 0.70 N.kg^{-1}. Again the experimental value 0.07 N.s.rad^{-1} exhibits a good learning performance but multiplying this value by 10 would prevent the agent to drive the pendulum to the target.

As mentioned in the experimental setup description, the real system has uncertainties associated to the measurement of the angle θ. In the virtual environment, this is accounted for by Gaussian noises of standard deviations σ_θ and $\sigma_\theta/\Delta t$ for the measurements of θ and $\dot{\theta}$ respectively. From the real system, we have evaluated $\sigma_\theta \sim 2.6$ mrad. Here we probe values ranging between 0 and 175 mrad in simulation (Fig. 2.15c). Low measurement noises, *i.e*, $\sigma_\theta < 8.7$ mrad, result in a perfect control quality as observed with high plateau values of the inference curves. A noise amplitude of 17.5 mrad is still acceptable. Beyond this value, the pendulum can not be driven to its unstable position.

Finally, we examine the effect of an associated degree of uncertainty on the command sent to the motor, thus a Gaussian noise of standard deviation σ_U is added to the voltage U in simulation. We show in Fig. 2.15d that, up to a noise level of $\sigma_U/U \simeq 0.1$, a good control is achieved. This condition is not restrictive and is easily obtained with classical systems. A moderate noise does not seem to impact the quality of the learning process.

CartPole Simulation analysis To verify the robustness of DQN training, we take 10 random seeds [5] [Henderson et al., 2018] on the training of a default configuration [Table A.1,Table A.2] and verify that all the results converge in Fig. 2.16. If not specified, we fix the seed number to 1.

5. random generator coefficient used for exploration and network weight initialization. To reproduce the results, same random generator coefficient should be used.

We will now compare the simulation performance of various deep RL algorithms, described in Section 2.3.3, with discrete and continuous actions. The hyperparameters list can be found in Appendix A.4.

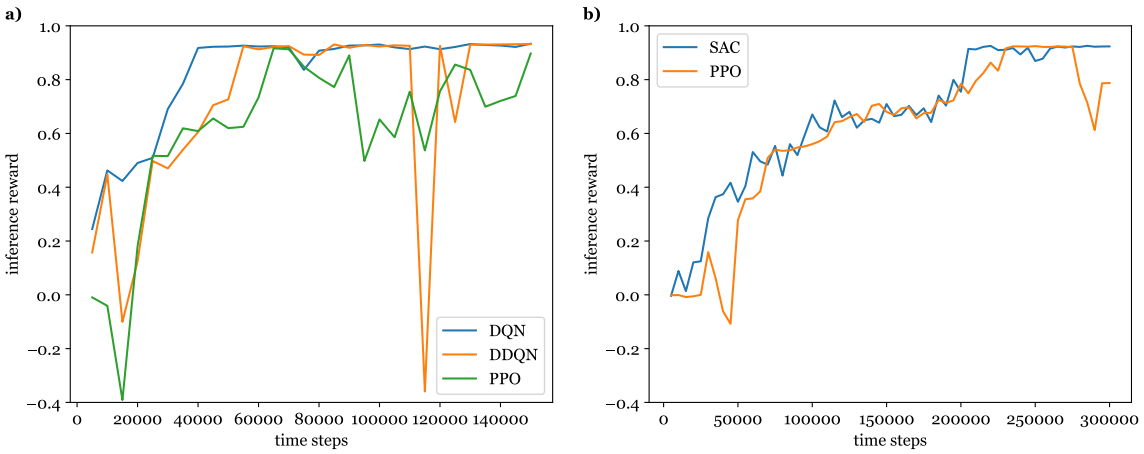

Figure 2.17 – a) Convergence of different RL algorithms on a **discrete**-action environment in inference. We notice that the algorithms reach almost the same maximum values during inference evaluations. To achieve better results for DDQN and PPO, the hyperparameters need to be more fine-tuned. b) Convergence of different RL algorithms on a **continuous**-action environment in inference. The training happens during 300000 time steps and it can be inferred that the algorithms takes longer to converge due to the continuous action space.

2.5 Conclusion

We have revisited in a pedagogical context, the stabilization of an inverted pendulum, a classical problem in dynamics and control theory. We introduced several model-based control techniques, a linear controller and Lyapunov-theory based controller. The Linear Quadratic Regulator which provides an optimal linear control when the pendulum comes near the unstable equilibrium, while Lyapunov theory permits to analyse the stability of nonlinear systems and design an appropriate controller. We re-derived the physical model of such a system and the control objective. Then, two model-free Reinforcement Learning algorithms were investigated both in experiments and in simulations, which offers an accurate description of real experiments. In terms of the control quality, the basic Q-Learning method is found not efficient while the more advanced algorithm DQN successfully accomplishes the stabilization of the pendulum in its unstable position, independently of the initial condition. Finally, we studied the influence of some extensive physical parameters on the control quality in simulation with the virtual environment. The robustness of the DQN approach has been therefore validated, both in terms of parameter influence, but also in terms of initial conditions : the RL always drives the pendulum in its unstable position, independently of the initial state. An admissible range of physical parameters were determined, which can be used to guide the elaboration of experimental setups.

Meanwhile, we deliberately chose to present results based on discrete actions for simplicity. We also verified the convergence of Soft Actor-Critic (SAC) algorithm [Haarnoja et al., 2019b] on a real cart-pole system as well. Using continuous action space with SAC unquestionably enables a finer control, but it takes more resources and approximately twice more interaction time to train the RL model due to additional complexity.

For public outreach, we provide all the details in an open-source code repository [Israilov et al., 2023], video of experiments with simulations [6] and the 3D model of the assembly [7] on the free-access web-platform. The cart-pole example sheds light on different control techniques that could be applied on a robotic fish which is a nonlinear underactuated system.

6. https://www.youtube.com/watch?v=XMn1FI9_f8k&ab_channel=SardorIsrailov

7. https://cad.onshape.com/documents/cb7bd0e195c60437cf97c9d5/w/e6da1cebe26dd182a7bf81c1/e/3d69146d5de92928caf63491?renderMode=0&uiState=64ef616331f16d113f004065

CHAPTER 3

Reinforcement learning approach to control a robotic fish

Undulatory swimming is the most observed mode of locomotion of underwater animals. In this chapter, we aim at determining strategies to achieve the highest possible thrust and consequently highest swimming speed. To address this challenge, we exploit a biomimetic robotic swimmer to determine the best command to produce the highest thrust and swimming speed. Using machine learning techniques, we uncover the optimal control which is rationalized using a simple model. In the first part of the chapter we focus on thrust optimization of a robotic fish both in simulation and in experiments. In the second half of the chapter, speed maximization is tackled in different simulation envirnments and a simple experimental setup is proposed to maximize the speed of a biomimmetic swimer using RL. We rationalise the results with corresponding discussions. Finally, a simple visual servoing task with a simulated fish virtual environment is presented, illustrating that RL is able to sucessfully drive the system from an initial to a target image.

3.1 Introduction and Context

The diversity of shapes and physiologies among multicellular organisms is tremendous, and it can be rationalized by the principles of Darwinian evolution and the various functions required by animals [Healy et al., 2019]. Locomotion is a vital activity for metazoans, as it enables them to fulfill essential functions necessary for survival, such as accessing favorable environments, engaging in reproduction, hunting, and evading predators. The swimming of underwater organisms [see Fig. 1.3] is usually split into two main groups : the smallest animals and the largest ones. The Reynolds number, which is the ratio of the fish velocity times the fish length to the fluid kinematic viscosity, is the discriminant parameter that permits the classification of the swimmers into the two categories [Childress, 1981]. For a small animal, i.e. with a Reynolds number lower than 1, the inertial effects can be disregarded and viscous friction controls the locomotion, whereas, for a larger animal, viscosity plays a minor role. In this chapter, we will focus on the gaits of the second category, and in particular, the undulatory locomotion which is the most observed form of swimming [Di Santo et al., 2021].

From tadpoles of a few centimeters to whales of 20 meters in length, swimming consists in pushing the water by undulating the body [Childress, 1981] which produces a thrust exploiting the inertia of the displaced fluid [Gazzola et al., 2014]. The kinematics of underwater undulatory swimming appear to be particularly robust in vertebrates, highlighting general physical principles. The wavelength of the body deformation is of the order of the length of the swimmer [Videler, 1993, Santo et al., 2021], while there is on average a factor 0.2 between tail beat amplitude and swimmer length [Bainbridge, 1958,Rohr and Fish, 2004,Hunter, 1971,Saadat et al., 2017,Sánchez-Rodríguez et al., 2023]. The tail beat frequency f is not fixed for an individual but tunes its swimming speed : the higher the frequency, the higher the speed [Bainbridge, 1958, Triantafyllou et al., 1993, Gazzola et al., 2014, Saadat et al., 2017, Sánchez-Rodríguez et al., 2023]. There is evidence that each swimmer can vary its frequency within a frequency band whose range is set by the interplay between the muscle properties and the interaction of the swimmer with its surrounding fluid [Sánchez-Rodríguez et al., 2023]. Muscles have their own limits in terms of speed of contraction and tension, as represented by Hill's muscle model [Hill, 1938]. Constraints can also be imposed by decision processes that are either spontaneous, through proprioceptive reflexes [Pearson, 1995, Williams IV et al., 2013, Sánchez-Rodríguez et al., 2021], or conscious, for instance through the choice of the activity level [Brett, 1964, Brett, 1972]. These decisions drive the gait [Di Santo et al., 2021] and we can expect different control strategies for an individual swimming at burst speed [Hirt et al., 2017] and the same swimmer exhibiting a swim-and-coast gait in a sustained level of activity [Li et al., 2021a].

These considerations are echoed in biomimetic robotic swimmers [Lebastard et al., 2016, Zhu et al., 2019, Sánchez-Rodríguez et al., 2021, Thandiackal et al., 2021, Lee et al., 2022]. The biological nature of the internal dynamics is here replaced by robotic elements : electronics, mechanics and/or computer science. Developing efficient and fully autonomous artificial swimmers requires finding the appropriate control strategies tailored to specific constraints to achieve optimal performance for a given action. Recent years have witnessed the development of various approaches to control soft robotic fish, both in simulations and experiments. Classical control techniques such as PI [Zhang et al., 2015], PID [Yu et al., 2004], and robust controllers [Zhang et al., 2016] have been employed to improve trajectory tracking performance. The emergence of artificial intelligence algorithms has paved the way for novel control approaches designed specifically for soft-material swimmers. Models and simulations are now utilized to explore various machine learning algo-

rithms, achieving high swimming speeds while maintaining trajectory accuracy [Novati et al., 2019, Jiao et al., 2021, Rajendran and Zhang, 2022, Youssef et al., 2022]. In one study, [Rajendran and Zhang, 2022] studied DDPG-based [Lillicrap et al., 2016] control of the simulated robotic fish to follow path and control yaw angle. In another, [Youssef et al., 2022] applied three different RL techniques (PPO, A2C, DQN) to experimentally guide a robotic fish towards two goals at the opposite ends of a water tank via camera, with predefined harmonic policy. Regardless of the methodology employed, the quest to achieve the highest swimming speed through robotic fish remains an enduring challenge in the domain of aquatic locomotion.

A number of simulation models have been created to study fish locomotion. In [Liu et al., 2022b], authors propose a set of physics-based environments for different modes of fish locomotion integrated in openai gym deep RL framework. In [Jiao et al., 2021], the authors implemented a PPO agent to drive a three-linked fish to a desired location in 2D in a potential flow, utilizing the sense of proprioception to reach the destination. In [Tassa et al., 2018], the authors present the general set of control benchmarks based on the popular physics-based simulation engine, mujoco [Todorov et al., 2012]. We present the modified "fish" environment from this control suite benchmark in Section 3.3.1 as well as the simulation results.

Furthermore, the study [Zhu et al., 2019] investigated the impact of frequency driving on the efficiency of a robotic swimmer in terms of thrust, speed and cost of transport. Another research [Epps et al., 2009] studied the swimming performance of fish-like robot where the servomotor is commanded by a square wave input signal but without justifying the choice of signal shape. While these investigations have explored the correlation between swimming speed and tail beat frequency, none of these prior studies has offered a conclusive comprehension of the optimal control strategy required to attain this objective.

Despite the high heterogeneity of morphology observed in large aquatic animals, only few quantities surprisingly characterize swimming kinematics. For example, the wavelength of the undulatory deformation λ appears to be almost constant and proportional to the length L of the animal : λ ranges from $0.7L$ to $1.1L$ for anguilliform to thuniform waving patterns, respectively [Videler, 1993, Santo et al., 2021].

In this chapter, we aim to optimize the robotic fish thrust force and swimming speed. First, we present our experimental setup for generating maximum thrust in swimming using a single-parameter control system. Then, we showcase the experimental outcomes achieved by employing deep RL techniques to drive a swimming robot [Sutton and Barto, 2018]. Through these experiments, we identify the optimal control strategy, which involves utilizing a square wave function that alternates between the two extreme values allowed by the controller. As we research insights on the underlying physical laws of aquatic locomotion, we need reliable and high-quality data coming from different sensors of a robotic fish. That's why we trained deep RL in two different training sessions while using two independent sources for learning the swimming gait of the fish [see Fig. 3.1, Fig. 3.2] : raw images from camera and force sensor readings. These experiments provided us with the multi-source validation of obtained swimming policy that couldn't be easily inferred from simple physical models.

To gain a deeper understanding of the underlying physics, we develop a theoretical model in Section 3.2.2, with the corresponding parameter identification in Section 3.2.2.1. This model supports and explains the observed experimental results. Furthermore, we justify the control strategy with the theoretical explanation using bang-bang control with the Pontryagin's Maximum Principle [Kirk, 2004]. Then, we present a simple model-free strategy "swinging", that only utilizes the angular velocity of a fin of the robotic fish to efficiently propel the robotic fish.

In the second half of the chapter, we proceed with the speed optimization of an autonomous fish in a water tank. We consider the problem of maximizing the speed of a moving robotic fish both in simulation and experiment. First, we describe the methods for finding optimal swimming gaits in different simulation environments. We propose a simple model of fish swimming in 1D with a visual simulation environment and find that the optimal control to be bang-bang. We also consider the problem of reaching a target speed in the minimum time. We then validate the square wave control with the physics-based simulation engine, mujoco [Todorov et al., 2012]. We also rigorously validate our optimized thrust strategy through a comprehensive 2D numerical simulation in COMSOL. We then proceed with the visual-servoing formulation of the problem that aims to steer the robotic fish from an initial image to a target image in minimum time. We present briefly the concept of Variational Auto-Encoder [Kingma and Welling, 2014a] and couple it with a RL algorithm to drive the robotic fish to the simulated target image in the optimum time. Once again, the optimum control is found to be bang-bang.

The experimental setup for learning an optimal swimming gait for speed maximization is described in Section 3.3.2. The RL learning of speed maximization in experiment further validates the square wave control strategy found before. By aligning experiments, theory and numerical simulations, we successfully determine the most efficient approach to achieve maximum swimming speed. This integrated approach provides valuable insights into the best methods for enhancing propulsion in aquatic systems.

3.2 Swimming thrust optimization

3.2.1 Experimental setup for thrust optimization

The robotic fish, designed in CAD software, consists of a rigid head made from *PLA* material and an elastic tail fabricated from *TPU* material. The robotic fish rear body consists of a deformable skeleton with a fin attached to its end, fabricated through 3D printing using a flexible polymer. To achieve controlled deformation, a servomotor is employed, connected to the skeleton's end via two cables passing through the holes in the fish tail. Its soft tail and caudal fin are made of a flexible polymer called "Ninjaflex" from NinjaTek. The tail with the caudal fin is printed with 20% of density in order to accommodate elasticity, while the rigid head is printed with 100%. The tail is actuated by a waterproof servomotor (Hitec HS-5086WP) that is located in a rigid head. The motor is connected to two nylon fishing cables actuating the elastic tail. The ensemble of an elastic tail, rigid head and servomotor is close to the neutral buoyancy in water. A Raspberry Pi 4B (8Gb) controls the servomotor using pulse-width modulated (PWM) signals with the 6V stable power source.

We utilize the experimental platform described above, which is equipped with the robotic fish illustrated in Fig. 3.1. By rotating the wheel of the servomotor by an angle $\phi(t)$, we can induce a deformation that controls the fin angle $\alpha(t)$. This design closely emulates the functioning of antagonistic muscles found in natural swimmers, responsible for initiating body deformations. The robot is immersed inside a water tank and its head is fixed to a force sensor that measures the longitudinal force $F_x(t)$. This force serves as a measure of thrust and is, on average, positive when the fish is in the propulsion phase. The setup of the robotic fish used in this study is illustrated in Fig. 3.1. We thoroughly present the main features of the setup; some applications and additional details can be found in [Gibouin et al., 2018, Sánchez-Rodríguez, 2021].

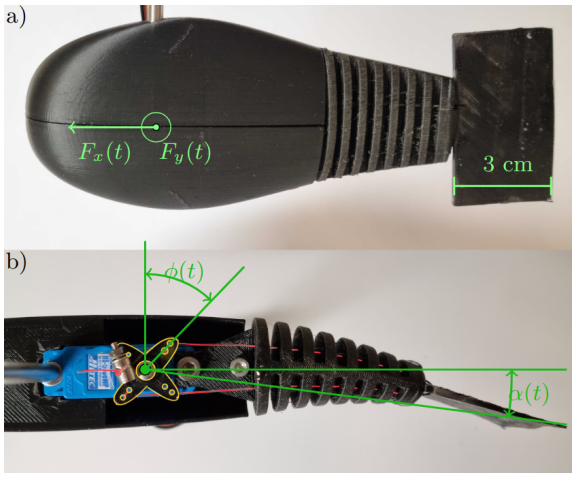

Figure 3.1 – Experimental setup used for thrust optimization. a) Side view of the robotic fish. b) Top view without the top fairing. The head is attached to a dual-force sensor. The tail and the fin are printed with a flexible polymer. The body deformation is triggered by the actuation of a waterproof servomotor (blue) : the rotation of its wheel pulls cables (drawn in red) to deform the elastic tail [Gibouin et al., 2018]. The contours of the servomotor wheel is drawn in yellow.

To measure forces, a force sensor (Honigmann RFSR®150EI) is linked to the robotic fish via an aluminum rod, enabling bi-directional force measurements (longitudinal force F_x and normal force F_y) for up to 5 N with a precision of approximately 10^{-3} N. The force sensor outputs the voltage linearly dependent on the force, and can be calibrated by applying the known force on it. The signals from the force sensor are filtered by a low-pass electronic filter at 20 Hz and amplified to fit the range of an ADC converter ([0, 5] V *i.e.* 1 N = 1 V). Then, an analog force signal is converted into a digital format using an ADC converter (Adafruit 1115) and collected by the Raspberry Pi via the I2C interface.

The robotic swimmer is positioned in a water tunnel (Rolling Hills Research Corporation, Model 0710) using a beam clamp that holds the aluminum rod. The water tunnel is described in Appendix D.1 and illustrated in Fig. D.1. In this experimental study, our focus is on determining the highest thrust generated by the oscillation of the swimmer's tail ; hence, no water flow was generated by the tunnel pump.

For the learning process based solely on force sensor measurements, we directly conducted it on the Raspberry Pi 4B (8Gb). The Raspberry Pi controls the servomotor using pulse-width modulation (PWM) signals. We used *pigpio* library for servomotor actuation and sensor management. The training of neural networks was accomplished using a machine learning framework *pytorch* installed on a microcontroller. However, for the learning process involving image-based observations, we utilized an external computer due to computational constraints of convolutional neural networks. Data exchange between the Raspberry Pi and the computer occurred via the TCP/IP protocol using an RJ45 Ethernet cable. We proceed with two learning methods before presenting the learning results in experiment.

Deep RL optimization with force sensor

In our pursuit of finding the best functional form for the servomotor driving, we employed deep RL techniques within the framework of Markov Decision Processes (MDP).

The state of the swimmer, denoted as $s = (F_y(t), \dot{F}_y(t), \phi_c(t))$, is characterized by the normal force $F_y(t)$ and its derivative $\dot{F}_y(t)$, as well as the command angle of the servomotor $\phi_c(t)$. The action, represented by $\phi_c(t) \in [-\Phi, \Phi]$, refers to the command angle of the servomotor. The reward is the average longitudinal force $\overline{F_x} = -\int_0^T F_x(t)dt$ over a time interval. The objective of the RL algorithm is to maximize the total cumulative reward, which, in this context, translates to maximizing the thrust $\overline{F_x}$ generated by the swimmer. The agent-environment interaction loop can be visualized in Fig. 3.2a.

To achieve this, the RL algorithm explores the available action and state spaces and gradually converges to the best control sequence through the trial and error process. We exploit the PPO (Proximal Policy Optimization) algorithm [Schulman et al., 2017] for this study for its ability to handle both discretized and continuous action spaces.

In our experiments, RL applies the control policy and sends a command to the robotic fish every 50 ms. The control policy is incrementally updated (trained) after each 768 control steps. We chose this number to ensure sufficient exploration of the action and state spaces before each update. The entire training process spans 10^5 control steps, which amounts to approximately 2 hours. Throughout this process, we save the control policy and the state value function every 5000 steps.

After each training of the neural networks, we conduct an inference to evaluate the quality of the actual best control policy : only the best action is chosen at each step during the episode without exploration. This evaluation provides valuable insights into the performance of the swimmer with the optimized control sequence. We now state the motivation and the method of using images as an input of RL algorithms.

Optimizing thrust from raw images

The use of images in the optimization process of machine learning algorithms has been considered hard due to the high dimensionality of images. Convolutional Neural Networks (CNN) were designed to extract usable features from high-dimensional images. We give a gentle overview of CNNs in Appendix B.1. Nowadays, embedded cameras have become affordable, embodying the source of rich information in the form of visual data. When observing robotic fish, high-dimensional images contain data about their motion and undulation of the tail due to the interaction with the fluid, which is hard to do with ordinary sensors. Coupled with Particle Image Velocimetry (PIV), cameras could also provide information about the fluid speed field surrounding the robotic fish.

In our study, we employed a web-camera (ODROID USB-CAM 720P) placed above the robotic fish's undulating fin to record its motion. When passing an image as an observation, the agent encounters the problem of partial observability of MDP, meaning that just one single image as an observation does not contain enough information to control the system with memory-free ANNs. Indeed, a single image may contain rich information about the deflection angle, but does not contain information about the speed which is crucial for robust feedback control. To tackle this challenge, several modifications can be applied. The use of RNNs or LSTMs [Hochreiter and

Schmidhuber, 1997] coupled with deep RL algorithms [Hausknecht and Stone, 2015, Chen et al., 2016] permit to alleviate the partial observability problem.

However, the straightforward way of addressing this challenge is to stack several consecutive images as an observation [Mnih et al., 2013, Mnih et al., 2015] to contain more information such as first and second derivative of the deflection angle. At each time step, four images are used (i.e., the current image and the previous three ones) [Mnih et al., 2013], to represent the state s of the robotic fish instead of the previous state representation $(F_y(t), \dot{F}_y(t), \phi_c(t))$. The recorded RGB images were preprocessed by converting them to gray-scale, re-scaling them from their original size to $(84, 84)$ pixels and normalizing to [0,1] range. The learning process can be visualized in Fig. 3.2b. This resizing was performed to reduce the compute time, while maintaining sufficient information. Value normalization ([0,1]) of input images being a standard practice in deep learning is performed to better train ANNs that rely on gradient descent algorithms. We used Convolutional Neural Networks (CNN), introduced by [LeCun et al., 1989, LeCun et al., 2015], as a feature extraction tool from rescaled images ($84 \times 84 \times 4$ pixels). The architecture was the same as in [Mnih et al., 2015]. The output of CNN is a latent 512 dimensional vector which is then passed to the fully-connected neural networks which constitute the actor-critic algorithm. Beyond this change in state representation, the remaining experimental methods for learning remained the same as those used for learning from the force sensor. We now present the results obtained in experiments and simulations.

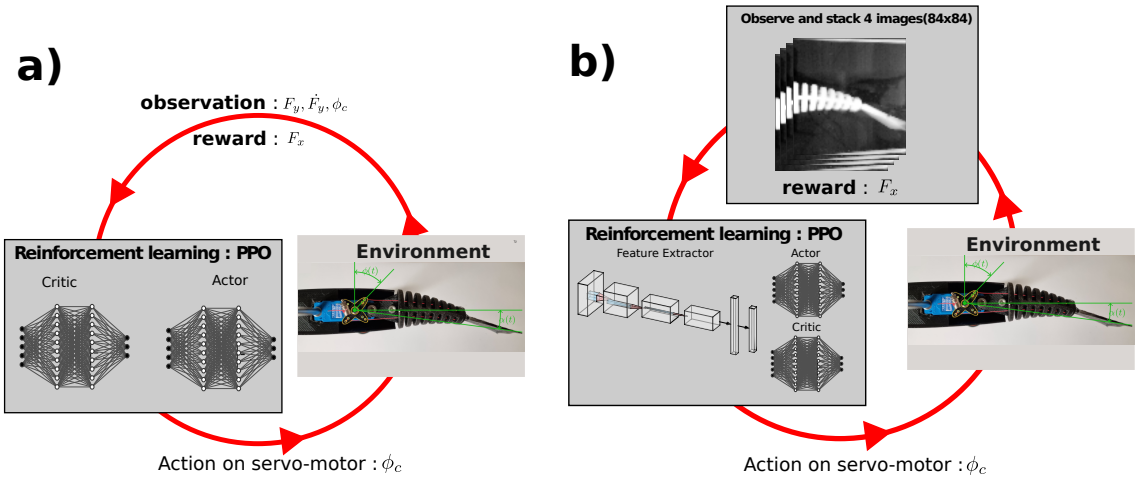

Figure 3.2 – Different modalities for RL training : a) Force sensor based observation : Observe normal force F_y, its derivative $\dot{F}_y$ and feed them to PPO algorithm together with the current consign ϕ_c and reward F_x. b) Image-Based observation : In this approach, four consecutive images from the environment are observed and stacked together. These stacked images are then processed through a Convolutional Neural Network (CNN) prior to being input to the PPO algorithm. The algorithm then generates an action consign ϕ_c to drive the servo with the elastic fin.

Results

Experiments and machine learning

We employ a continuous-action PPO algorithm to identify the optimal control signal that maximizes the average thrust. As stated before, we limit the instruction angle $\phi_c(t)$ of the servomotor to vary within the interval $[-\Phi, \Phi]$ and the objective is to maximize a reward proportional to the thrust $F_x(t)$. The algorithm requires input observations to characterize the state of the system at every time step *i.e.* 50 ms. Two independent sets of inputs are consecutively considered to probe the robustness of the learning process and ensure the convergence toward the optimal control signal.

First, the algorithm is provided with $F_y(t)$ and its time derivative $\dot{F}_y(t)$ to account for the oscillatory nature of the system. After a few hours, the learning process converges toward a periodical motion. As shown in Fig. 3.3, the optimal control signal $\phi_c^\star(t)$ consists of a square wave function that switches abruptly between the two allowed extreme values $\pm\Phi$.

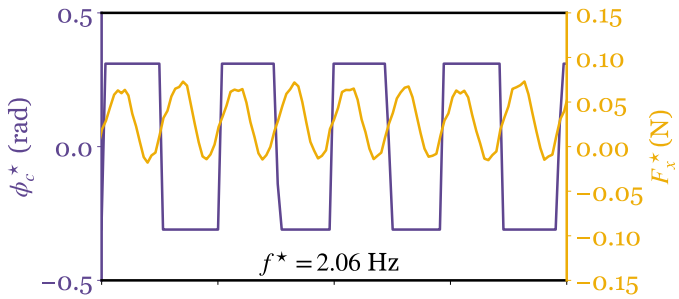

Figure 3.3 – Experimental result : instruction angle $\phi_c^\star(t)$ and $F_x^\star(t)$ of the best strategy revealed by the algorithm, for $\Phi = 0.31$ rad.

Second, we provide images recorded by a camera, as shown in Fig. 3.1b, as inputs instead of relying on lateral force measurements [see Section 3.2.1]. Despite this change in inputs, the resulting optimal control signal remains consistent with the previous findings. Once again, the algorithm converges to a square wave function with the same amplitude and frequency, reaffirming the robustness of the previously identified optimal control strategy. The video of a final policy can be visualized here [1] and the video comprising input images to the feature extractor, here [2].

The two approaches are compared in terms of relevant physical quantities while varying the maximum instruction angle Φ. In Fig. 3.4, the maximized average thrust $\overline{F_x}^\star$, the optimal frequency $f^\star$ and the corresponding maximum fin angle $\alpha_{\max}^\star$ are plotted as functions of Φ. Whatever Φ, the agreement is excellent between the two approaches for all quantities. Specifically, $\alpha_{\max}^\star$ is proportional to Φ; the greater Φ, the greater the thrust, as one would expect from the gain in new possibilities as Φ increases; the optimal frequency is a slowly decreasing function of Φ and varies between 2.5 and 1 Hz in the parameter range.

To confirm that square wave forcing produces the greatest thrust, we prescribed three types of classical periodic forcing (sinusoidal, triangular, and square wave) for a given Φ and measured the averaged thrust as a function of forcing frequency f. Results are provided in Fig. 3.5 for $\Phi = 0.42$

1. https://youtu.be/npo2xaA5ao0
2. https://youtube.com/shorts/_mSvU12Uo8c

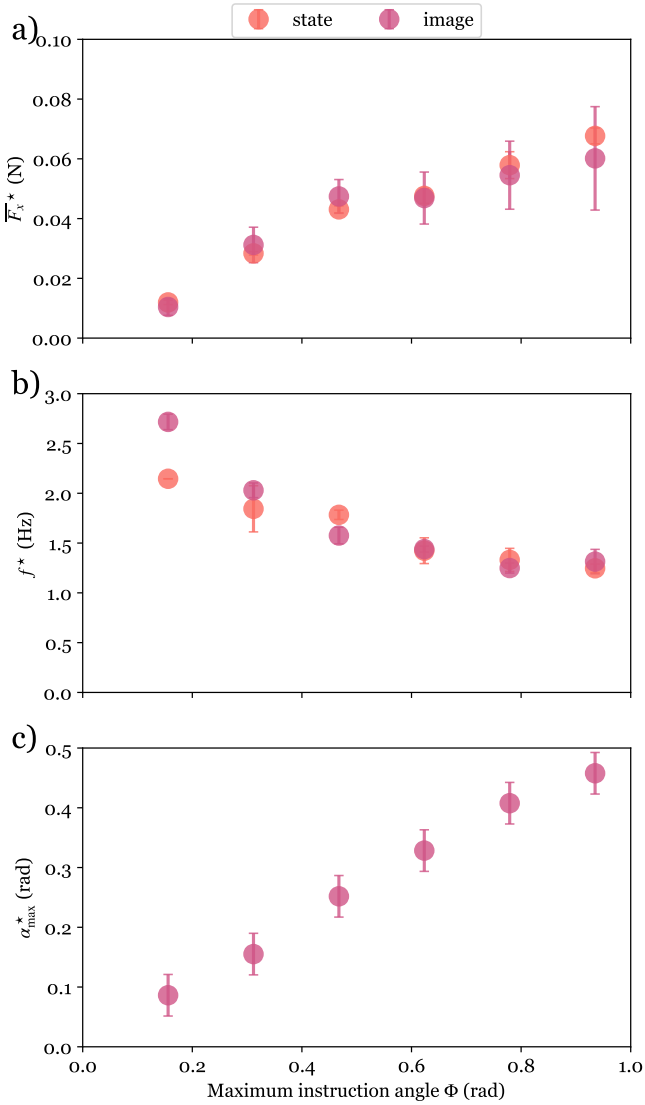

Figure 3.4 – a) Maximum average thrust $\overline{F_x^\star}$. b) Optimal undulation frequency $f^\star$. c) Maximal fin angle $\alpha^\star_{\max}$. For each figure, we compare the experimental results from image observations and those obtained with the state observations.

rad. Regardless of the type of actuation, the variation of the forcing frequency demonstrates the presence of a peak in thrust. Remarkably, square wave forcing consistently generates the highest average thrust. This maximum thrust is achieved at a frequency of 1.8 Hz, a value that closely aligns with the frequency obtained using the RL approach.

The analysis reveals a robust feature of optimal control : a square wave function oscillating between two extreme values. The control frequency, around a few Hz, aligns with the frequency observed in natural swimmers of similar size [Sánchez-Rodríguez et al., 2023]. This suggests the existence of a strong mechanism driving maximum thrust.

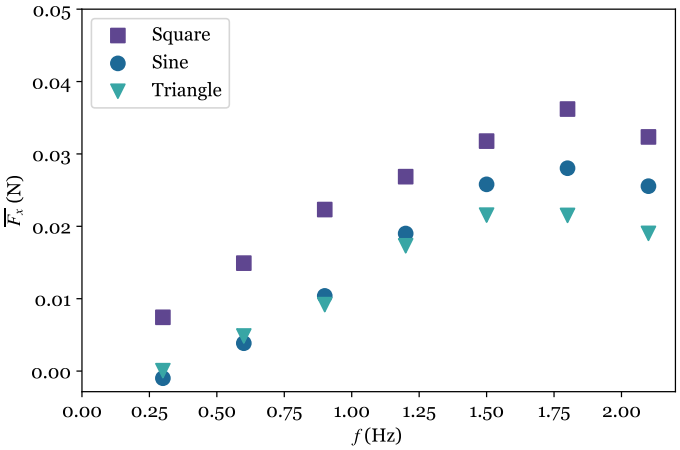

Figure 3.5 – Experimental results : influence of the frequency on the thrust produced by some periodic control policies : square wave (purple squares), sinusoidal (blue circles) and triangular (green triangles) waves. $\Phi = 0.42$ rad.

3.2.2 Biomimetic robotic fish model

We present a comprehensive model that combines the physical aspects of underwater undulatory swimming with the internal dynamics of the servomotor to gain insights into the optimal solutions of a swimming gait.

The dynamics of the fin angle $\alpha(t)$ is described as that of a damped harmonic oscillator, influenced by the angle $\phi(t)$ of the servomotor wheel [Sánchez-Rodríguez et al., 2021] :

$$\ddot{\alpha}(t) + \xi\omega_0\dot{\alpha}(t) + \omega_0^2(\alpha(t) - \alpha_c(t)) = 0,\ \alpha_c(t) = \lambda\phi(t). \tag{3.1}$$

This equation captures the interaction between the deformable fin and the surrounding water, driven by an instruction $\alpha_c(t)$ proportional to the servomotor wheel angle $\phi(t)$. The numerical values of the parameters were determined following the procedure described in Section 3.2.2.1. The proportionality factor $\lambda = 0.46$ depends on cable length and polymeric skeleton elasticity. Additionally, the servomotor has its own internal dynamics to adjust the servomotor wheel angle $\phi(t)$ to the instruction angle $\phi_c(t)$ [Sánchez-Rodríguez et al., 2021] :

$$\dot{\phi}(t) = \Omega \tanh\left(\frac{1}{\Delta}\left(\phi_c(t) - \phi(t)\right)\right). \tag{3.2}$$

In Appendix C, we justify this model of servo-motor from first principles and physics derivations. The model includes the parameters $\Omega = 5.8$ rad.s^{-1} and $\Delta = 0.29$ rad, which respectively represent the maximum angular speed of the wheel and the required angle difference for the servomotor to operate at its maximum angular speed. The nonlinearity introduced by the $\tanh$ function results in the saturation of the wheel speed $\dot{\phi}(t)$ to $\pm\Omega$ if the servomotor is unable to keep up with the provided instruction. This saturation occurs when the servomotor is too slow to follow the desired angle $\phi_c(t)$. Eqs. (3.1) and (3.2) elucidate two essential aspects of the dynamics : one pertains to the interaction between the undulating swimmer and its aquatic environment, while the other accounts for the limitations imposed by the swimmer's internal dynamics.

Our experimental system serves as a suitable candidate for mimicking the dual nature of dynamics observed in natural fish. In natural fish, the internal dynamics are limited by biological constraints at the muscular level [Sánchez-Rodríguez et al., 2023].

The thrust is proportional to the mass of water accelerated in the longitudinal direction and is written $F_x = -K\alpha(t)\ddot{\alpha}(t)$ [Gazzola et al., 2015, Sandha et al., 2021, Sánchez-Rodríguez et al., 2021], where the parameter $K = 12.9\ 10^{-3}$ N.rad^{-2}.s^2 characterizes the thrust efficiency of the robot in water and is measured following the procedure described in Section 3.2.2.1. The average thrust over an undulation period $T = 1/f$ can be written as :

$$\overline{F_x} = -\frac{K}{T}\int_0^T \alpha(t)\ddot{\alpha}(t)dt = \frac{K}{T}\int_0^T \dot{\alpha}(t)^2 dt. \tag{3.3}$$

We employ PPO on the model (Eqs. (3.1) and (3.2)) to verify the optimal control strategy that yields maximum average thrust. In Fig. 3.6b, we show the time evolution of the optimal instruction angle $\phi_c^\star(t)$ for $\Phi = 0.31$ rad, alongside the corresponding rescaled fin angle $\alpha^\star(t)/\lambda$ and thrust $F_x^\star(t)$. Consistent with the experimental results, the optimal instruction angle adopts a square wave function, alternating between Φ and $-\Phi$.

In fact, the dynamics presented in Figs. 3.6a and b, measured in experiments and simulated with the model, are very similar. This is confirmed by the excellent agreements in Fig. 3.7b and Fig. 3.7c for the relevant quantities such as undulation frequency and maximum fin angle of the

optimal control. Furthermore, this simple model capably reproduces the average thrust force, as depicted in Fig. 3.7a. Regardless of the specific value of Φ, both the experiments and the model demonstrate that the system waits for the fin angle to approach its instruction before changing the control. In other words, the control signal switches direction when the fin angle is close to its maximal value $\alpha^\star_{\max} = \lambda\Phi$. This finding suggests an intuitive mechanism for selecting the frequency simultaneously with the control switch, emphasizing the efficiency and adaptability of the system in achieving maximum thrust.

The model is also validated with the data obtained in Fig. 3.8 while enforcing sinusoidal, triangular and square wave forms. The agreement between the experiments and the model is again excellent. The maximum thrust is generated around a frequency of 1.8 Hz, indicating that the system operates optimally near its resonance in $\dot{\alpha}$, which occurs at a frequency $\omega_0/(2\pi) \sim 2.0$ Hz, regardless of the damping factor. As a comparison, the resonance in α suggested by various studies [Michelin and Llewellyn Smith, 2009, Paraz et al., 2016, Hoover et al., 2018] occurs at a lower value, $\omega_0\sqrt{1-\xi^2/2}/(2\pi) \simeq 1.1$ Hz in our system. We now briefly explain how the parameters of the robotic fish model were identified in practice.

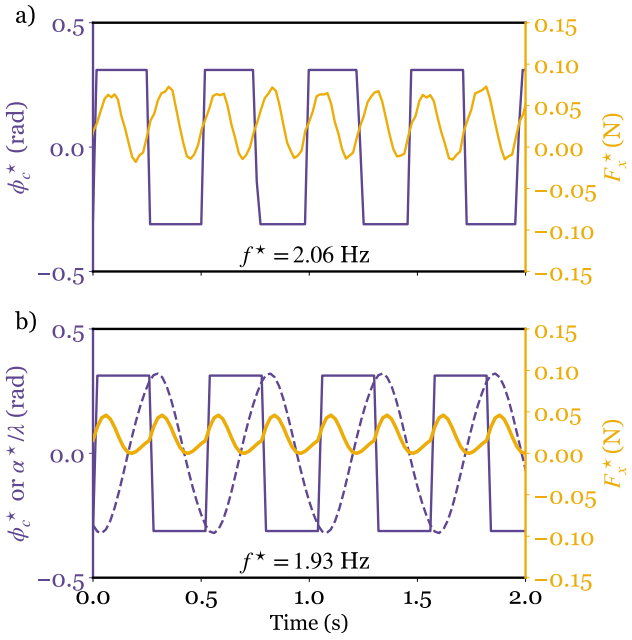

Figure 3.6 – Comparison between experimental and simulation results : a) Experimental result : instruction angle $\phi_c^\star(t)$ and $F_x^\star(t)$ of the best strategy revealed by the algorithm, for $\Phi = 0.31$ rad. b) Temporal evolution of the instruction $\phi_c^\star(t)$ (purple solid line), the rescaled fin angle $\alpha^\star/\lambda$ (purple dashed line) and the resulting thrust $F_x^\star(t)$ (orange solid line), obtained for optimal control with the model and the parameters $\Omega = 5.8$ rad.s^{-1}, $\Delta = 0.29$ rad, $\omega_0 = 12.5$ rad.s^{-1}, $\xi = 1.2$ and $\Phi = 0.31$ rad.

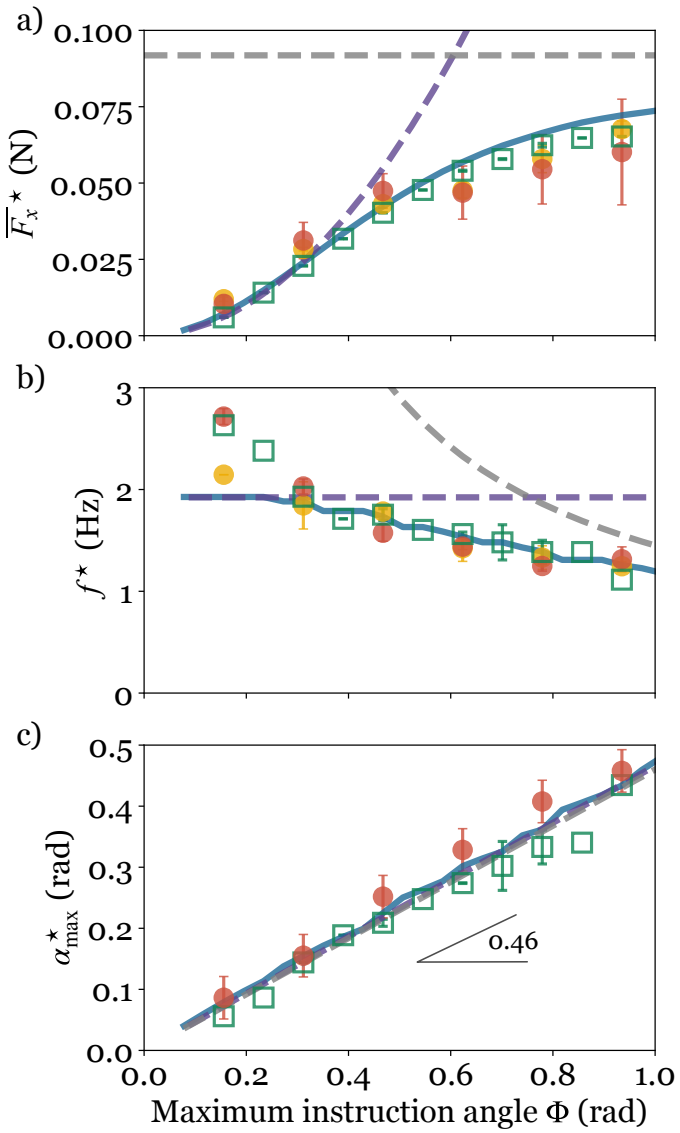

Figure 3.7 – a) Maximum average thrust $\overline{F_x^\star}$. b) Optimal undulation frequency $f^\star$. c) Maximal fin angle $\alpha^\star_{\max}$. For each figure, we compare the experimental results and those obtained with the model (Eqs. 3.1, 3.2 and 3.3). Experimental results : the red symbols are obtained via RL with the state $(F_y, \dot{F}_y, \phi)$; the orange symbols are obtained via image learning, where the state is a set of four successive pictures of the robot. Simulation results : the open square symbols denote RL results with the simulation of the model. The dashed violet and gray curves represent theoretically predicted values in the limits of very small and very large Φ, respectively. The solid green curve corresponds to the numerical determination of the frequency $f^\star$ yielding the highest thrust $F_x^\star$ assuming a square waveform for ϕ_c.

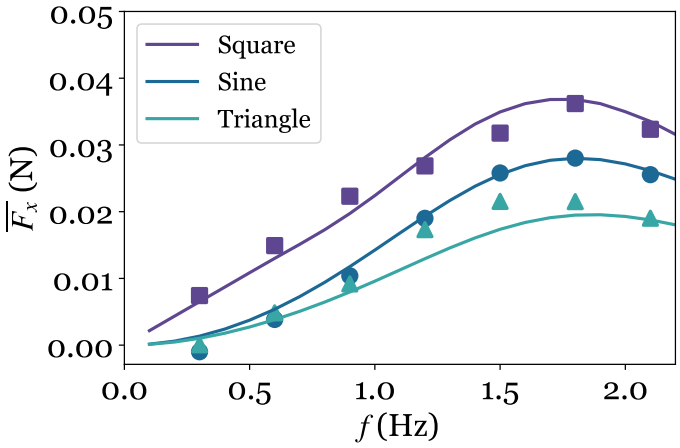

Figure 3.8 – Influence of the frequency on the thrust produced by some periodic functions : square wave (purple squares), sinusoidal (blue circles) and triangular (green triangles) waves. Filled symbols and solid lines correspond to the experimental points and the model predictions, respectively. $\Phi = 0.42$ rad.

3.2.2.1 Parameter identification

The process of determining parameters through fitting is carried out using both linear regression and nonlinear fitting techniques, specifically employing the *lmfit*[3] library for the latter. Initially, the robotic fish is actuated using different control angles : $\phi_c = 20°, 40°, 50°$ and at various frequencies in the interval $f = [0.2, 1.2]$. The procedure is divided into following steps :

1. The corresponding maximum fin deflection angle $\alpha^\star_{\max}$ is measured from the videos recorded of fin oscillating at these different angle amplitudes and frequencies [see Fig. 3.10].
2. The proportionality factor (between α and ϕ) $\lambda = 0.46$ is determined in quasi-static experiments *i.e.* for experiments conducted at low frequency square wave forcing.
3. From [Sánchez-Rodríguez et al., 2021], the mean thrust force $\overline{F_x}$ is proportional to $\omega^2 \alpha^{\star\,2}_{\max}$. We employ ridge linear regression to fit the coefficient K in Eq. (3.3), setting $\overline{F_x}$ as a function of $\omega^2 \alpha^{\star\,2}_{\max}$ [see Fig. 3.9b].
4. With a pre-defined range for physically plausible values of different parameters, we proceed with nonlinear fit of the parameters Ω, Δ of Eq. (3.2) and ξ, ω_0 of Eq. (3.1) on the experimental data. The result of the fit can be visualized in Fig. 3.9b.

We have demonstrated with different methods both in physical simulations and experiments that the optimal gate is square wave control. The found control function also known as bang-bang control arises as an optimal control of many dynamical systems, especially the dynamical system guided by linear equations. We now proceed with the theoretical explanation when bang-bang control is optimal.

3. `https://lmfit.github.io/lmfit-py/`

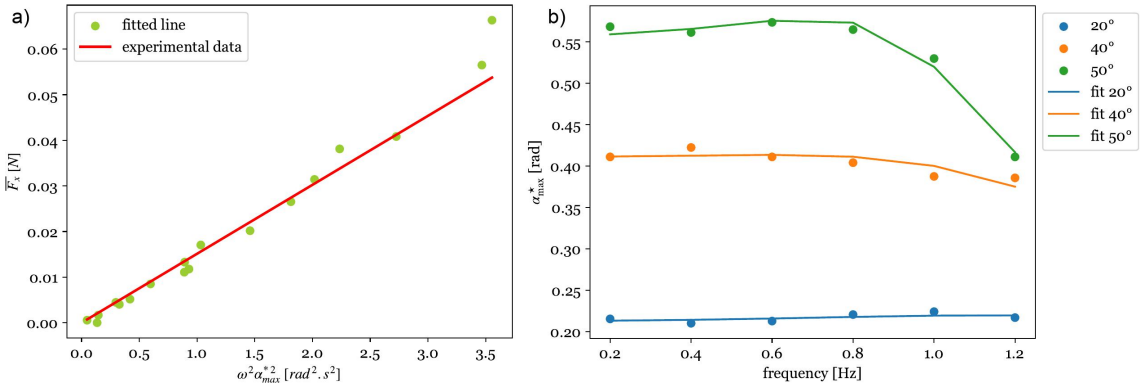

Figure 3.9 – Parameters fitting results comparison with actual data. a) Fit of parameter $K = 12.9\ 10^{-3}$ N.rad^{-2}.s^2 in equation Eq. (3.3) via linear regression b) Superposition of measured angle of the fish fin undulation and prediction by the model. The data from three servo-motor actuation angles ($\phi_c = 20°, 40°, 50°$) and 6 different frequencies ($f \in [0.2, 1.2]$) were used to find the parameters $\Omega = 5.8$ rad.s^{-1}, $\Delta = 0.29$ rad, $\omega_0 = 12.5$ rad.s^{-1} and $\xi = 1.2$.

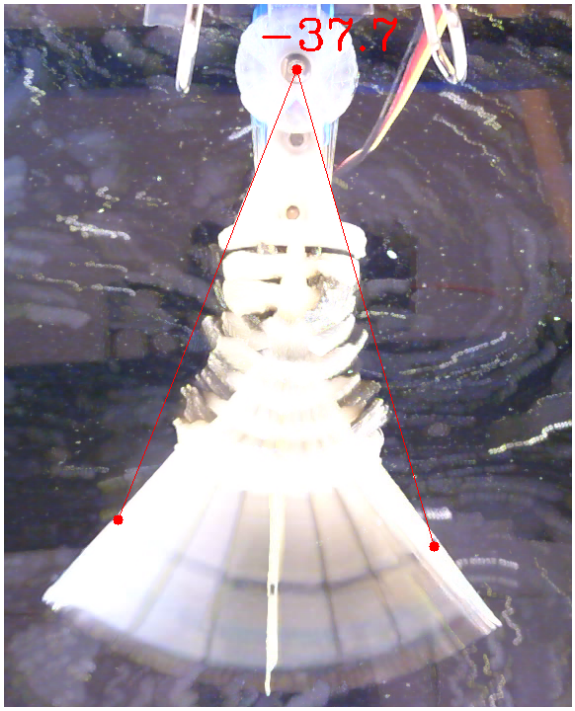

Figure 3.10 – Mean image obtained from averaging of images from video recording of several tail undulations. Example of determining two times $\alpha^{\star}_{\max}$ from mean image. The angle is measured from the static marker near the end of caudal fin.

3.2.3 Bang bang control

Bang-bang control, also known as on-off control, is a control strategy that switches between two extreme values of control under certain criteria [Sonneborn and Van Vleck, 1964]. The criteria of change can be a certain threshold of a physical quantity or an error signal for the control. The bang-bang controller provides the optimal control if the response is driven by linear equations [Kirk, 2004]. The necessary condition and the switching between the upper and lower bounds of control derives from the Pontryagin's Maximum Principle [Pontryagin, 1987] and yields that the optimal minimum-time control has a bang-bang structure. Below, we give a simple introduction to a bang-bang controller based on Pontryagin's principle.

3.2.3.1 Pontryagin's Maximum Principle

Let a dynamic system described by :

$$\dot{\mathbf{X}}(t) = f(\mathbf{X}(t), \mathbf{U}(t)),$$

where :

— $\mathbf{X}(t)$ is the state vector, $(x_i(t) \quad i = 1, \ldots, n)$.
— $\mathbf{U}(t)$ is the control vector, $(u_i(t) \quad i = 1, \ldots, m)$.
— f is the system dynamics.

The objective of the optimization is to minimize a cost function at each point in time $[0, T]$:

$$J = \int_0^T I(\mathbf{X}(t), \mathbf{U}(t))dt,$$

where $I(\mathbf{X}(t), \mathbf{U}(t))$ is the running cost. The Hamiltonian, in this context, is defined by :

$$H(\mathbf{X}(t), \mathbf{U}(t), \lambda(t)) = I(\mathbf{X}(t), \mathbf{U}(t)) + \lambda^T(t)\dot{\mathbf{X}}(t)$$

Hamiltonian optimizes the objective function $I(\mathbf{X}(t), \mathbf{U}(t))$ with $\lambda^T(t)$ called *costate variables* that depend on time, unlike Lagrange multipliers used for static constrained optimization. Hamiltonian can be viewed as a modified Lagrangian in the constrained optimization problem. Hamiltonian is obtained via the Legendre transformation of a Lagrangian.

Pontryagin's Maximum Principle states that in order to maximize the objective, the optimal state trajectory $\mathbf{X}^*(t)$, optimal control $\mathbf{U}^*(t)$, and corresponding Lagrange costate vector $\lambda^*(t)$ must maximize the Hamiltonian H so that :

$$H\left(\mathbf{X}^*(t), \mathbf{U}^*(t), \lambda^*(t)\right) \geq H(\mathbf{X}(t), \mathbf{U}(t), \lambda(t))$$

or equivalently :

$$\mathbf{U}^* = \operatorname{argmax} H\left(\mathbf{X}^*(t), \mathbf{U}(t), \lambda^*(t)\right), \tag{3.4}$$

for all time $t \in [0, T]$ and for all permissible control inputs $\mathbf{U}(t) \in \mathcal{U}$ ($\mathcal{U}$ is a control domain). Moreover, the corresponding boundary conditions must hold. Here, the trajectory of the costate vector $\lambda(t)$ is the solution to the costate equation and its terminal conditions :

$$\begin{aligned}
&\frac{\partial H(\cdot)}{\partial u_k} = 0, \quad k = 1, \ldots, m \\
&\frac{\partial H(\cdot)}{\partial x_i} = -\dot{\lambda}_i(t), \quad i = 1, \ldots, n \\
&\lambda_i(T) \geq 0, \quad \lambda_i(T)x_i^*(T) = 0, \quad i = 1, \ldots, n.
\end{aligned}$$

It can be shown that bang-bang is optimal for a linear control with a linear cost on time, while LQR can be derived using Pontryagin's Maximum Principle when the optimized cost is quadratic on state and action.

For example, let us study a linear problem : we want to drive the dynamical system from the initial state $\mathbf{X}(0) = \mathbf{X}_0$ to the final state $\mathbf{X}(T) = 0$. Let the system dynamics and optimization cost function be defined as :

$$\begin{aligned}\dot{\mathbf{X}}(t) &= A\mathbf{X}(t) + B\mathbf{U}(t), \quad |\mathbf{U}(t)| \leq 1\\ J &= \int_0^T 1dt,\end{aligned}$$

To find the optimal control, we form the Hamiltonian

$$H(t) = 1 - \lambda^T(t)(A\mathbf{X}(t) + B\mathbf{U}(t)) = 1 - (\lambda^T(t)A)\mathbf{X}(t) - (\lambda^T(t)B)\mathbf{U}(t).$$

The application of Pontryagin's Maximum Principle yields the following :

$$\begin{aligned}\dot{\mathbf{X}}(t) &= \frac{\partial H(t)}{\partial \lambda} = A\mathbf{X}(t) + B\mathbf{U}(t)\\ -\dot{\lambda}(t) &= \frac{\partial H(t)}{\partial \mathbf{X}} = A^T\lambda(t)\\ \mathbf{U}(t) &= \arg\max H(t) = -\operatorname{sgn}(\lambda^T(t)B)\end{aligned}$$

It follows that the input is always $\mathbf{U} = \pm 1 \Longrightarrow$ "bang-bang" in order to extremize the Hamiltonian. In Appendix A.7, we present the cart-pole stabilization problem seen before using bang-bang control, to clarify the concepts. In the next section, we use Pontryagin's Maximum Principle for deriving an optimal control of a robotic fish.

3.2.3.2 Bang-bang controller for fish thrust maximization

In the context of the model with Eqs. (3.1) and (3.2), we apply Pontryagin's Maximum Principle [Pontryagin, 1987] to explain why square wave control at maximum allowed values $\pm\Phi$ in the instruction angle ϕ_c produces the maximum thrust $\overline{F_x}$, defined in Eq. (3.3), under the constraints of Eq. (3.1) and Eq. (3.2).

The determination of the maximum thrust thus requires the resolution of a variational problem subject to three Lagrange multipliers, associated with the dynamic equations for $\alpha(t), \dot{\alpha}(t)$ and $\phi(t)$. This condition is not completely satisfied in this system due to the nonlinear behavior of the servomotor's internal dynamics, Eq. (3.2). However, a bang-bang controller remains the optimal choice in this system because of the specific nature of the nonlinear function driving the relaxation dynamics of ϕ (Appendix E.2).

This appears particularly simple to explain for a fast servomotor, i.e., if $\phi(t)$ almost immediately follows the command $\phi_c(t)$. We perform a dimensionless analysis to evaluate the rapidity of the servomotor. We make the following change of variables :

$$s = \omega_0 t \tag{3.5}$$

$$\alpha(t) = \lambda\Phi\beta(s) \tag{3.6}$$

$$\phi(t) = \Phi\psi(s), \tag{3.7}$$

and substituting in Eqs. (3.1) and (3.2) we get the system to solve :

$$\ddot{\beta}(s) + \xi\dot{\beta}(s) + \beta(s) - \psi(s) = 0 \tag{3.8}$$

$$\dot{\psi}(s) = \frac{\Omega}{\Phi\omega_0}\tanh\left(\frac{\Phi}{\Delta}(\psi_c(s) - \psi(s))\right), \tag{3.9}$$

which uncovers the three dimensionless parameters that govern the dynamics.

— The parameter ξ determines the Q factor of the oscillator in $\alpha(t)$ or $\beta(s)$.

— The parameter $\frac{\Omega}{\Phi\omega_0}$ controls the capacity of the servomotor to perfectly follow the command $\psi_c(s)$. If $\frac{\Omega}{\Phi\omega_0} \gg 1$, we can eliminate adiabatically the dynamics of $\psi(s)$: in this limit, we replace the Eq. (3.2) for the wheel angle ϕ by $\phi(t) = \phi_c(t)$. This corresponds to the definition of a fast servomotor.

— The parameter Φ/Δ measures the nonlinear response of the servomotor.

  — if $\Phi/\Delta \gg 1$, the function $\tanh$, can be replaced by the function sign and because of the relaxational dynamics, the servomotor equation is reduced to

$$\dot{\phi}(t) = \Omega\mathrm{sign}\phi_c(t) \tag{3.10}$$

  — if $\Phi/\Delta \ll 1$, the equation in ψ can be linearized, and we can write the equation :

$$\dot{\psi}(s) = \frac{\Omega}{\omega_0\Delta}\left(\psi_c(s) - \psi(s)\right), \tag{3.11}$$

  or in dimension variables :

$$\dot{\phi}(t) = \frac{\Omega}{\Delta}\left(\phi_c(t) - \phi(t)\right). \tag{3.12}$$

  Hence, the servomotor will perfectly follow the command if $\frac{\Omega}{\omega_0\Delta} \gg 1$.

In summary, the servomotor can be considered as fast in two regimes : first if $\frac{\Omega}{\Phi\omega_0} \gg 1$, second if $\frac{\Omega}{\Delta\omega_0} \gg 1$ and $\Phi/\Delta \ll 1$. In our case, we can assimilate $\phi(t)$ to $\phi_c(t)$ for $\Phi \ll \Phi_s$ (from the first regime), with :

$$\Phi_s = \frac{\Omega}{\omega_0}. \tag{3.13}$$

In both limits, the forcing command acts linearly through the Eq. (3.1), which completely justifies the choice of a bang-bang controller as an optimal strategy. In addition, this asymptotic permits the computation of the average thrust resulting from a square wave driving at frequency f :

$$\overline{F_x} = K\lambda^2\Phi^2\omega_0^2\frac{\frac{4f}{\xi\omega_0}\left(\sinh\left(\frac{\xi\omega_0}{4f}\right) - \frac{\xi\sinh\left(\sqrt{\xi^2-4}\frac{\omega_0}{4f}\right)}{\sqrt{\xi^2-4}}\right)}{\cosh\left(\sqrt{\xi^2-4}\frac{\omega_0}{4f}\right) + \cosh\left(\frac{\xi\omega_0}{4f}\right)}, \tag{3.14}$$

where $K\lambda^2\Phi^2\omega_0^2$ is the dimensional value that gives the scaling of the average thrust in this limit. Regardless of the value of ξ between 0 and 2, this thrust is maximum when $\dot{\alpha}$ is resonant and the oscillator is driven close to its undamped frequency : $f^\star \simeq \omega_0/(2\pi)$ (see Appendix E.2).

For the case of a slow servomotor with $\Phi \gg \Phi_s$, the system of equations is nonlinear, and the maximum thrust can be expressed as :

$$\overline{F_x} = K\lambda^2\Omega^2, \tag{3.15}$$

which is reached as long as the servomotor wheel is moving at the maximum angular speed Ω (i.e. $\dot{\alpha}(t) = \pm\lambda\Omega$ in Eq. (3.3)). The square wave forcing at maximum allowed values appears as the optimal solution to maximize the difference between $\phi_c(t)$ and $\phi(t)$ in Eq. (3.2) and ensures that the servomotor wheel is moving at maximum angular speed. In this limit, the optimal undulation period is determined by the shortest time required to sweep the servomotor angle from $-\Phi$ to Φ and then back to $-\Phi$, all at maximum angular speed. This leads to the expression $f^\star = \Omega/(4\Phi)$.

The behavior resulting from both limits is depicted in Fig. 3.4, and agrees with the outcomes obtained in experiments and with the model, as well as indicating a transition for Φ around Φ_s.

Swinging control : a model-free strategy

In the above model, to maximize thrust, the optimal frequency must be found in order to select the best square wave function for control. It is therefore necessary to perform preliminary calibrations to measure the relevant quantities, in this case ω_0, ξ, Ω and Δ, to bring a particular system closer to optimality. Here we explore another strategy that does not require any prior knowledge about the system.

Starting from the thrust expression, Eq. (3.3), the optimization is intrinsically linked to the time evolution of the fin angle velocity $\dot{\alpha}$. An intuitive approach would be to favor the phases of maximum speed by reversing the sign of the instruction angle when the fin slows down too much. We call this swinging control in reference to the way children are able to swing without knowing the physical laws involved.

To this end, we introduced an instruction-changing criterion C ($0 \leq C \leq 1$), such that ϕ_c changes sign if the speed slows to $C\dot{\alpha}_M$, with $\dot{\alpha}_M$ the last observed maximal angle speed. This condition ensures that the highest possible speed is maintained and that the instruction is changed when the speed becomes too slow. We found numerically that this strategy is always independent of the system history and the initial conditions, as the algorithm converges toward a unique solution.

We evaluated the average thrust resulting from the swinging strategy by varying the parameters $\omega_0\Phi/\Omega$, which measures the servomotor's ability to follow the instruction, and C. Fig. 3.11 shows that a criterion value of C ranging from 0 to 0.8 yields satisfactory performance for the swinging strategy whatever the value of $\omega_0\Phi/\Omega$. The swinging control consistently delivers excellent outcomes without imposing stringent constraints on the choice of C : the thrust efficiency, defined as $\overline{F^{\text{swing}}}/\overline{F^\star}$, being higher than 70% in the parameter range : for $C = 0.6$, this ratio exceeded 95% regardless of the value of $\omega_0\Phi/\Omega$, but fine-tuning remains possible. This strategy can be proposed even in specific cases : in the example of a fast servomotor ($\Phi \ll \Phi_s$) and an undamped oscillator ($\xi \to 0$), the optimum is attained when the instruction angle changes sign at $\dot{\alpha} = 0$ (i.e., $C = 0$) with a thrust efficiency very close to 100%, as shown in the inset of Fig. 3.11, and detailed in (Appendix E.2).

Swinging control is a robust strategy for achieving the highest thrusts without prior knowledge of the system or complex control algorithms. Its straightforward implementation using basic sensors makes it practical for various applications.

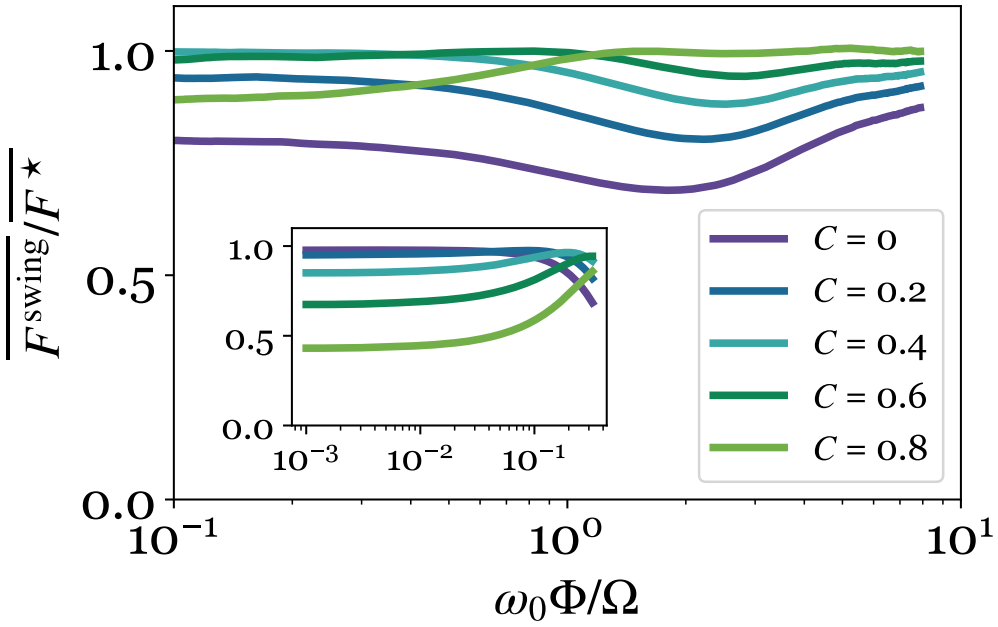

Figure 3.11 – Performance of the swinging strategies compared to the optimal solutions as a function of $\Phi/\Phi_s = \omega_0\Phi/\Omega$, with different instruction-changing criteria. We fixed $\omega_0 = 12.5$ rad.s^{-1}, $\xi = 1.2$, $\Omega = 5.8$ rad.s^{-1} and $\Delta = 0.29$ rad. Inset : Performance of the swinging strategies with the same instruction-changing criteria for a fast servomotor $\omega_0\Phi/\Omega < 1$ and $\xi = 0.05$.

3.3 Swimming speed optimization

Generated thrust force of the robotic fish is equilibrated by the drag force when certain speed is reached [see [Gazzola et al., 2014, Berg et al., 2021]]. Maximizing the thrust force is strongly correlated to maximizing the speed in practice. In this section, we propose a protocol for teaching a robotic fish to maximize the swimming speed while swimming against the direction of water flow. We first explore different existing simulators for fish swimming and propose to construct the simulation environment that resembles the dynamics of an experimental setup to understand the basic challenges and tune RL hyperparameters on a similar problem. We test the swimming speed optimization in three different simulation environments, each of them having their own advantages :

1. Custom defined dynamics and visualization built on top of openai gym framework [Brockman et al., 2016, Schulman et al., 2015]. The main advantages of this simulation lie in its simplicity. The simulation dynamics is clearly understood and can be easily derived from first principles.
2. Open-source physics-based simulation engine [Todorov et al., 2012, Todorov, 2014] that has API with python, using the benchmark simulation library. [Tassa et al., 2018, Tunyasuvunakool et al., 2020]. The simulation engine is open-source and is widely used in the reinforcement learning community. Although it's not precise for fluid dynamics simulation, it gives insights on approximate solutions for speed maximization of a fish immersed in water. The simulation can also be extended to 2D and 3D.
3. Comsol simulation which is a general-purpose simulation software based on advanced numerical methods. It is the most precise simulator among the three using computational fluid dynamics (CFD). CFD permits to analyse more precisely the flow and vortices induced by a robotic fish undulating its tail.

Simulation in the above virtual environments helps to understand the basic challenges of optimization and to find the expected solution for maximizing the swimming speed. After validation of the RL approach in simulation, we proceed with describing experimental setup and speed optimization procedure. We employed an experimental configuration similar to the one utilized in previous sections [see Fig. 3.1]. We discuss the obtained results and compare them to thrust optimization.

3.3.1 Learning to swim in a virtual environment

We hereby aim to optimize the robotic fish swimming speed in a single direction. The creation of a simulation that represents the dynamics of the experimental system used in experiments is a crucial part of designing an efficient controller, before deploying on a real system. As the simulation model is not known, we aim to approximate it with simple equations that describe the essence of physical phenomena present in the dynamical system.

Simulation model of fish swimming In the simulated environment, the equations governing the motion dynamics of the robotic fish and the signification of physical variables are derived from models of fish locomotion described in [Sánchez-Rodríguez et al., 2020, Sánchez-Rodríguez et al., 2021]. The dynamics of the fish tail flapping is modeled as a damped harmonic oscillator described before in Eq. (3.1). We assume for simplicity that the servo-motor actuating the fish's fin has fast dynamics [see Section 3.2.3.2] *i.e.* $\Phi \ll \frac{\Omega}{\omega_0}$, and ϕ almost instantly follows ϕ_c. Therefore, the

agent's command translates from ϕ_c directly to $\alpha_c = \lambda\phi_c$. The state is updated through equations of motion linking fish fin undulation $\alpha(t)$ and linear displacement $x(t)$ in the direction of thrust force :

$$\alpha_c(t) = \lambda\phi \quad \text{and } \phi = \phi_c \tag{3.16}$$

$$\ddot{\alpha}(t) + \xi\omega_0\dot{\alpha}(t) + \omega_0^2\left(\alpha(t) - \alpha_c(t)\right) = 0 \tag{3.17}$$

$$\ddot{x} = (-C_d|\dot{x}|\dot{x} - C_t\ddot{\alpha}\alpha)/m, \tag{3.18}$$

where Eq. (3.18) represents the equation governing the fish's linear displacement in one dimension with $-C_t\ddot{\alpha}\alpha$ being the thrust force, $-C_d|\dot{x}|\dot{x}$ being the drag force and m the unit mass. The thrust force coefficient C_t is calculated via the procedure described in Section 3.2.2.1, while the drag coefficient C_d was fitted from the second order polynomial fit described in Appendix D.1.3. In what follows, the results are presented in dimensionless units.

We aim to optimize the swimming speed of a simulated robotic fish governed by dynamics defined in Eq. (3.17) and Eq. (3.18). Here, we distinguish between two distinct types of observable states that can be employed. The first category includes the actual true sensor state, denoted by the vector $[x, \dot{x}, \alpha, \dot{\alpha}]$. Alternatively, the second category comprises the high-dimensional visual input of the simulator [see Fig. 3.12] with the same underlying dynamics. The simple visual input of a robotic fish is represented in Fig. 3.12. For the front-end visualization, we used gym [4] and pyglet [5] libraries.

The state of the system is updated using backward Euler integration scheme with a sampling time step of 20 ms. The derivation and significance of the above physical variables are available in [Sánchez-Rodríguez et al., 2020]. We use the hyperparameters listed in Table B.1. For the physical parameters of the simulation we refer to Table B.4. We used $1e^5$ steps for learning from true state observations and $1e^5$ for learning from images. All the following learning results in simulation are based on the continuous action space $\alpha_c \in [-\lambda\Phi, \lambda\Phi] = [-60°, 60°]$.

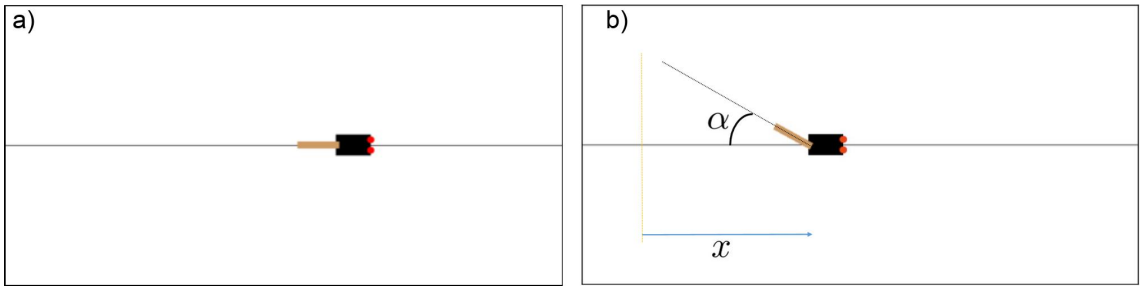

Figure 3.12 – Visualization from the simulator with size (400, 800). The thin brown rectangle represents the tail, the black rectangle with two red circles represents the fish body. For visual control, this input is sampled every sampling time and is then rescaled to (84, 84). a) raw image b) annotated for visualization purposes of x and α.

Fish locomotion training in simulation from true state

In our simulations, we applied PPO [Schulman et al., 2017] on the virtual environment with the true state given by $[x, \dot{x}, \alpha, \dot{\alpha}]$ and reward by $\dot{x}$. PPO was able to converge to the optimal policy

4. https://www.gymlibrary.dev/index.html
5. https://pyglet.org/

rapidly [see Fig. 3.13a]. The resulting swimming gait in Fig. 3.14 is bang-bang as initially expected. The hyperparameter list for PPO training can be found in Table B.1. To confirm the results, we also employed several predetermined control policies on a simulated robotic fish for comparison with the optimally derived swimming gait from RL [see Fig. 3.15]. Namely, sinusoidal, triangular and square wave swimming gaits were enforced as a predefined control policy during one episode and the swimming speed was recorded. The forcing simulation results follow a similar trend as before : square wave forcing outperforms sinusoidal, and sinusoidal is better than triangular forcing.

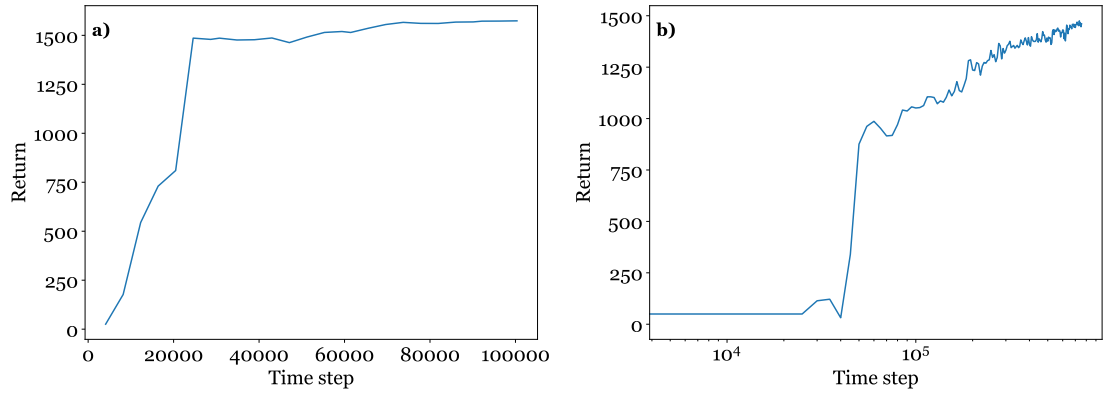

Figure 3.13 – Evaluation curve of learning via a) PPO algorithm trained on true state $[x, \dot{x}, \alpha, \dot{\alpha}]$ b) DrQv2 algorithm trained on processed images of a simulator. The curve represents the algorithm's evaluation score every 5000 time steps.

Learning to maximize the speed from visual input in simulation

The alternative to learning from true state $[x, \dot{x}, \alpha, \dot{\alpha}]$ is to learn from the visual data of a simulator with the same underlying dynamics. At every time step several consecutive processed images from previous time steps (see the original image in Fig. 3.12) are stacked with each other to form an observation to the agent.

The normal PPO with CNN feature extractor failed to optimize the speed from visual input in the reasonable time and we applied one of the State-of-the-Art model-free RL algorithms for visual control, DrQ-v2 [Yarats et al., 2021]. The algorithm DrQ-v2 consists of underlying DDPG [Lillicrap et al., 2016] algorithm with different data augmentation techniques and update schemes that significantly accelerate sample efficiency of the algorithm. The core of data augmentation in this algorithm consists of random cropping with subsequent rescaling of the input image instead of ordinary downsizing to (84, 84) pixels and augmenting the replay buffer with these augmented image transitions. We employed this algorithm for the control of fish swimming with continuous actions from visual input [see the video [6]]. Training from visual data gave comparable results as training from true state. The learning curve can be visualized in Fig. 3.13. The learning is stopped, yet the learning return is still increasing because learning rate is low when learning from image observations. The resulting superposition of episodic return in inference through true state and visual data as well as the enforced swimming gaits can be visualized in Fig. 3.15.

6. https://youtube.com/shorts/rMHhURZ_8z0

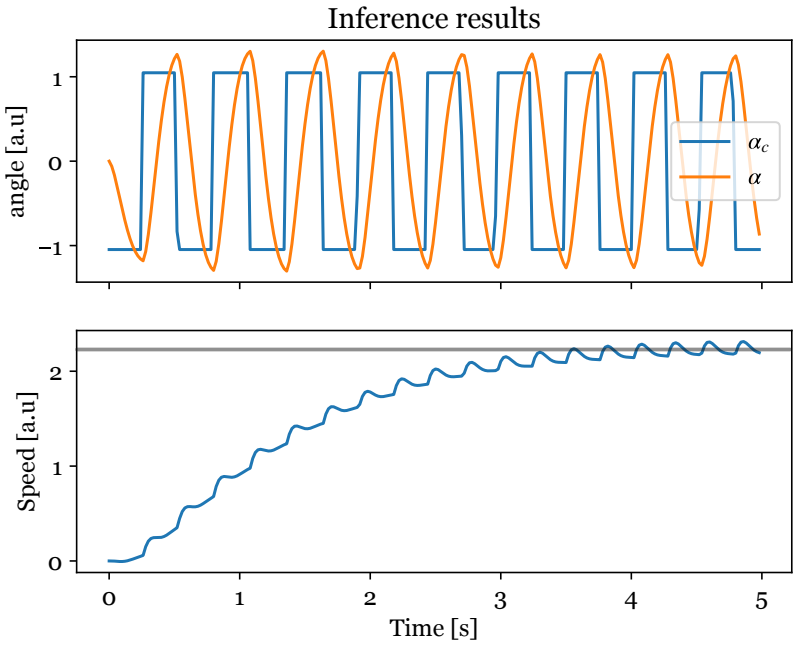

Figure 3.14 – Inference results with the best policy (a.u in labels correspond to adimensional units). Top : temporal evolution of the instruction $\alpha_c(t)$ (blue line) and the real angle $\alpha(t)$ (orange line), bottom : $\dot{x}$ evolution. The results are obtained for optimal control with the model and the parameters $\omega_0 = 11.8$ rad.s^{-1}, $\xi = 1.13$, $\alpha_c^\star = 1$ rad (maximum value), $C_d = 0.254$ and $C_t = 12.9$ mN.rad^{-1}.s^2.

Discussion We have seen that learning from true state can be substantially faster and yeild slightly better performance. When using stationary environment as with learning to maximize thrust force from images, we observed that learning from images is fast. This can be rationalized due to the relatively static composition of the environment which facilitate learning from images as the observation space is way smaller than the environment with the moving fish.

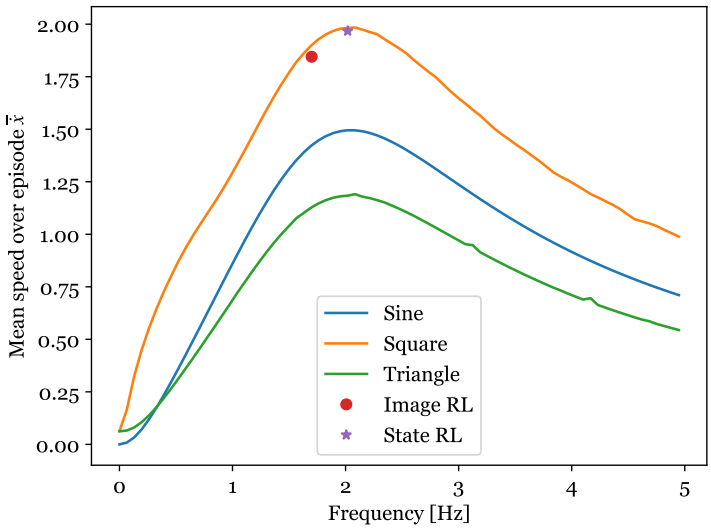

Figure 3.15 – Influence of frequency on episodic return (mean speed during 1 episode) in a custom fish swimming environment. The enforced swimming gaits consist of predefined functions : square wave (orange), sinusoidal (blue) and triangular (green) functions with frequency range $f \in [0.1, 5]$ Hz. The red circle is the learning results from DrQv2 on the visual data (PPO could not learn from visual data in a reasonable time). The star signifies the best inference return from the PPO training on the true state $[x, \dot{x}, \alpha, \dot{\alpha}]$.

Learning to achieve a target speed from true state

From simulations, we observe that the maximum achievable speed is more than 2 [a.u]. We now change the objective to achieve some target speed at a value of 1. For this, we adapt several quantities :

1. The reward at each time step to be $|\dot{x} - 1|$ instead of $\dot{x}$.
2. The observation is augmented with second derivatives and is now defined as $[x, \dot{x}, \alpha, \dot{\alpha}, \ddot{x}, \ddot{\alpha}]$. The second derivatives are added to encompass the necessary information about the current inertia of the system. Depending on this information controller is able to maintain the system at constant speed without oscillations.
3. The hyperparameter corresponding to the entropy weight in the loss function is taken to be much smaller. This is due to the fact that that there are many possible solutions to the desired speed, thus the entropy can take large values in this case and should not be accounted as much as before [see the hyperparameters in Table B.1].

The obtained results are visualized in Fig. 3.16 with learning statistics in Fig. 3.17. It can be clearly seen that the algorithm achieves the objective speed [dashed line in Fig. 3.16a] and changes the frequency of control when the speed is near the target, but is still square wave function.

The obtained result is not the coherent with natural fish that have the amplitude constant and change frequency depending of the aimed speed. The resulting control actions have a bang-bang tendency which we think is because the square-wave function possesses the high spectral entropy compared to other functions. There are many other control solutions to accomplish the speed servoing task.

As it can be seen from Fig. 3.17a, the main learning happens before 10^6 time steps when the evaluation return drastically increases and this can also be confirmed with loss drastically decreasing in this period [see Fig. 3.17b] corresponding to learning the value function. Increasing the learning rate of PPO improves the speed of convergence while sacrificing final performance.

While the method works well in simulation, learning this objective in real experiments incorporates several challenges. First, training for several million time steps on an experimental setup is too long to be feasible. More sample-efficient algorithm should be used instead. One of such, DROQ [Hiraoka et al., 2021], a variant of SAC incorporating dropout [Srivastava et al., 2014] and layer normalization [Ba et al., 2016] regularizations applied on critics, is able to learn accomplishing the task during 10^5 time steps [see Appendix B.4]. Another challenge arising in experiments in that the speed is in 2D, thus additional constraints should be taken into account. Furthermore, measuring an instantaneous robotic fish speed is subject to measurement noise, a moving average during $1s$ should be used instead.

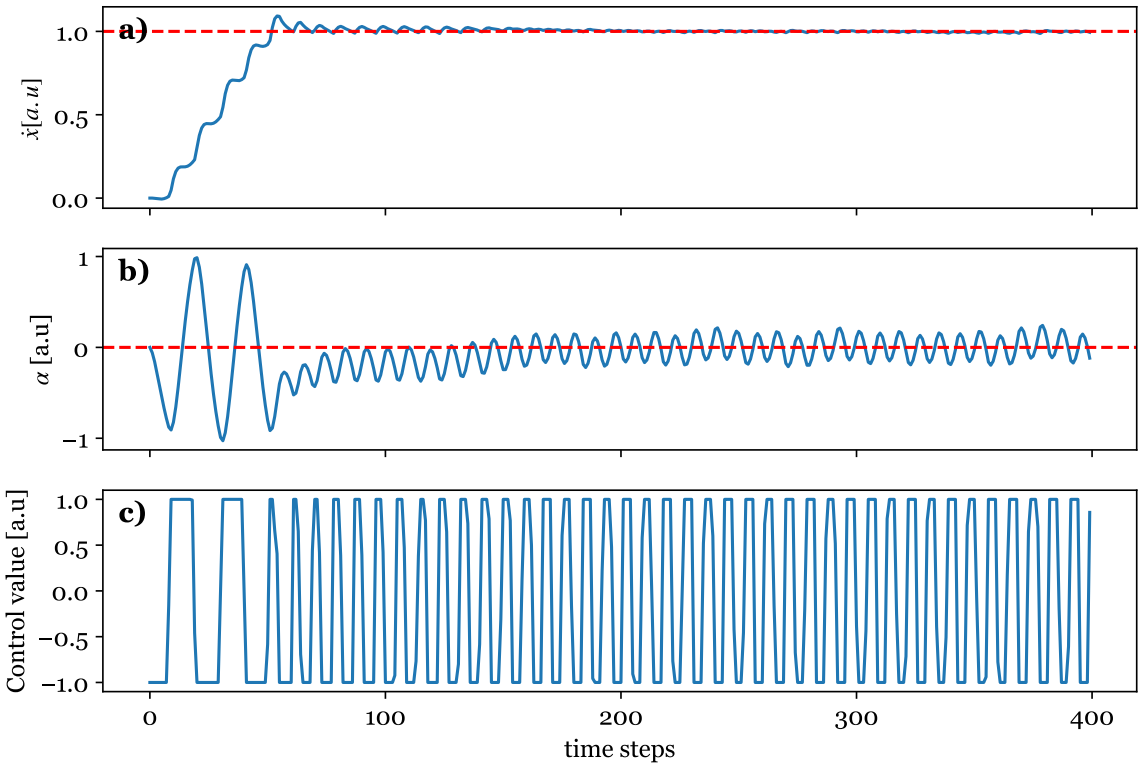

Figure 3.16 – Inference results from the best model found during learning with PPO. a) Speed $\dot{x}$ evolution during the first 400 time steps. We observe that the objective is perfectly accomplished. b) Fin angle evolution through time. First 50 time steps are related to the high amplitude undulations, corresponding to high produced thrust. This is coherent with Fig. 3.16a, when the fish need high thrust to achieve goal speed as quickly as possible. This is a simple model and for this reason α is not symmetric. c) Action evolution through time. Medium frequency of forcing corresponding to high torque is substituted with high frequency undulations when the objective is achieved.

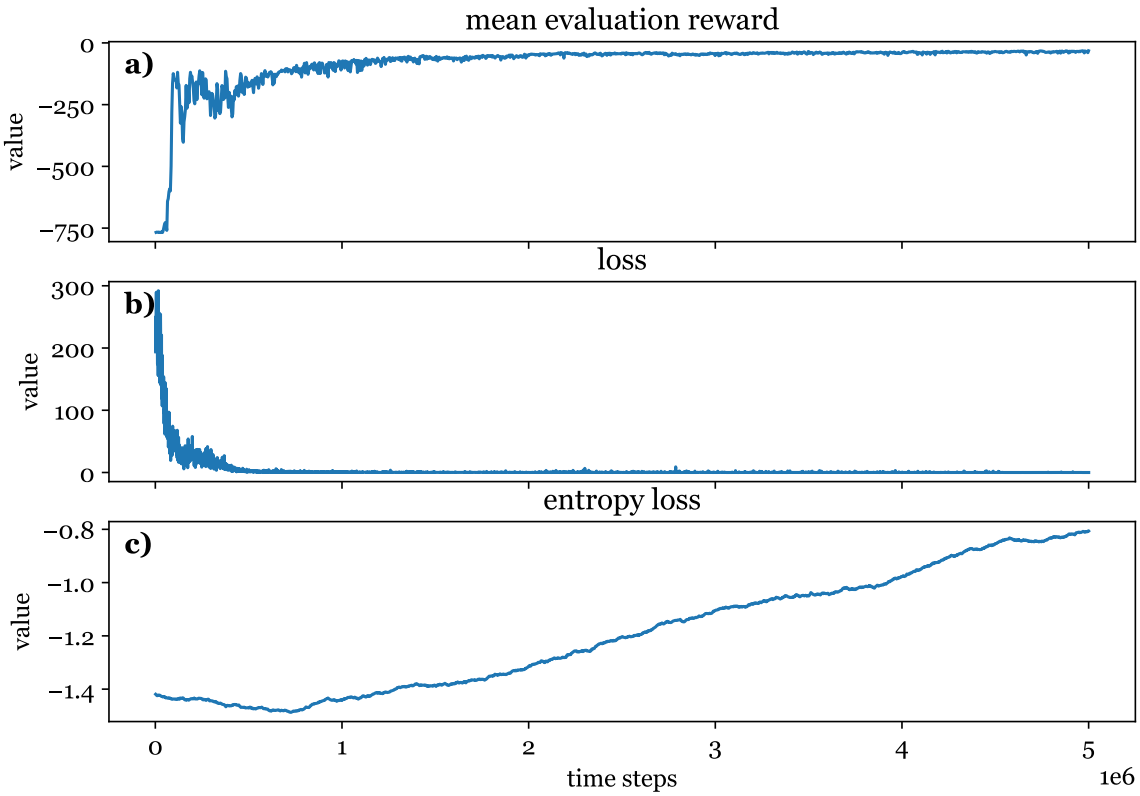

Figure 3.17 – Learning statistics for achieving target speed. a) Evaluation (inference) reward happening after every 5 episodes. We can see that the return starts with the value around -768, which corresponds to the number of time steps in an episode. This can be rationalized with the fact that at the beginning the policy does not do anything and the agent is penalized with -1 every time step. b) Loss evolution through time. Optimized loss decreases drastically before 500000 time steps. This is connected to learning the value function by the critic estimating whether the state has an estimated high return or not. c) Entropy loss (negative entropy) evolution through time. It decreases at the beginning signifying that the algorithm explores by increasing entropy of actions. Then, when an appropriate policy is found the entropy goes down to certain value indicating convergence to certain policy.

Speed controller with deep RL

Deep RL is applicable not only for extremizing specific quantities, but also for regulating these quantities to maintain a specific target value that can vary in real time. For example, in [Kaufmann et al., 2023] researchers used deep RL to guide the drone on a particular track, while in [Kim et al., 2004], the RC helicopter followed a particular shape in the air.

We adapt the objective of the previous task to maintain a variable target value, thereby rendering a deep RL agent as a speed controller. Our goal is to construct a learning task that enables an algorithm to adapt the policy to particular target velocity in real-time. For this, we adapt several quantities :

1. The observation space is defined as $[\dot{x}, \alpha, \dot{\alpha}, \ddot{x}, \ddot{\alpha}, \dot{x}_{target}]$
2. $\dot{x}_{target} \in [0, 2]$ changes at every episode and reward remains $|\dot{x} - \dot{x}_{target}|$ at every time step.
3. The initial conditions of the episode at the reset of the environment are taken randomly : $\dot{x} \in [0, 2], \alpha \in [-1, 1], \dot{\alpha} \in [-1, 1]$. As before, every episode lasts 768 time steps. This facilitates the generalization of an RL algorithm.

We used DROQ [Hiraoka et al., 2021] for continuous control of the system. During evaluation we impose different speeds for every 300 time steps [see red Fig. 3.18a]. From Fig. 3.18, we observe that the algorithm is able to follow the varying target speed perfectly. The agent adapts the corresponding amplitude of undulations [Fig. 3.18b] to produce the thrust force matching the drag force at a certain speed.

We now proceed with another simulation environment based on the popular physics-based simulation engine, in order to gain more insights into fish locomotion.

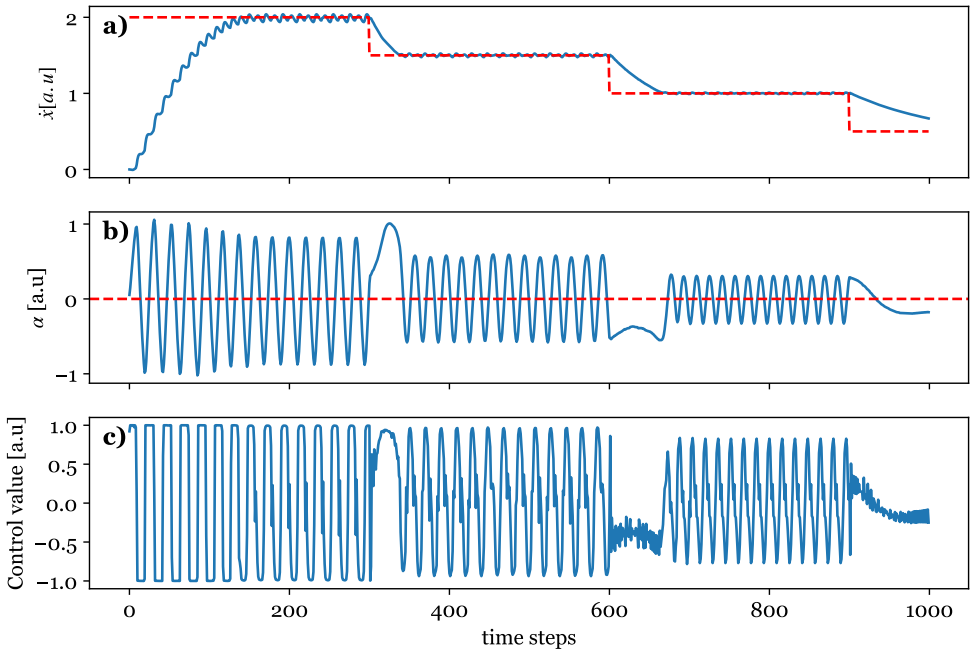

Figure 3.18 – Inference results for achieving different target speeds. a) $\dot{x}$ evolution during 1000 time steps with the consign (red dashed line). We observe that the objective is perfectly accomplished. b) Fin angle evolution through time. First 100 time steps correspond to high amplitude undulations, corresponding to high produced thrust. This is coherent with Fig. 3.18a, when the fish need high thrust to achieve goal speed as quickly as possible. c) Action evolution through time. During the changes in speed at 300 and 600 time steps, the angle control does not undulate. The algorithm waits until the drag force slows down the fish, and then the algorithm adapts the amplitude of oscillation to match the target speed.

Speed optimization in physics-based simulation engine – Mujoco

Mujoco (Multi-Joint dynamics with Contact) [Todorov et al., 2012, Todorov, 2014] is a physics engine designed for the simulation and control of robots and biomechanical models. It is especially popular in the reinforcement learning community for benchmarking algorithms in tasks that involve continuous control, like robot locomotion. Mujoco is a powerful open-source engine that incorporates friction, elastic elements and tendons. Although, commercial and finite elements analysis simulators like COMSOL are more precise than mujoco, simulations in mujoco remain computationally faster with user-friendly API [7] that is suitable for reinforcement learning experiments.

Many different systems have been modeled using mujoco engine, and we base our simulation environment on the fish environment from the "DeepMind Control Suite" [Tassa et al., 2018] set of environments. The modified version can be visualized in Fig. 3.19. The fish is a rigid body with a two-link tail. One part of the tail is actuated and another is compliant. The environment is modified to be constrained in 1D and restarts at the same point in space when an episode reaches 1000 time steps. The sample video can be visualized in the video [8]. We aim to maximize the speed in one direction using visual data. For the visual input to an RL algorithm, we take the image with the tracking camera mounted above the fish at each sampling time [see Fig. 3.19a].

In order to find an optimal swimming gait, we employed DrQ-v2 algorithm to find an optimal gait for mean speed maximization. The algorithm used the processed images from the top tracking camera as an input observation and actuated the fish fin with continuous actions $\phi_c \in [-\Phi, \Phi]$, where $\Phi = 60^\circ$ is a maximum deflection angle of an active caudal fin. The resulting swimming gait from learning is square wave. To verify the optimal gait is square wave, we applied three types of classical periodic forcing (sinusoidal, triangular and square wave) on ϕ_c and measured the averaged speed as a function of forcing frequency. The maximum mean speed is achieved at a frequency of 4.5 Hz (note that we can not compare with the results of the robot fish we presented before as the physical parameters are not the same). The results are shown in Fig. 3.20. The environment is defined by an *.xml* and *.py* files and the code of the environment with an algorithm is available here [9]. We now proceed with more precise simulator that performs CFD (computational fluid dynamics).

7. Application Programming Interface
8. `https://youtube.com/shorts/6jlbEY4By2c`
9. `https://github.com/ss555/drqv2`

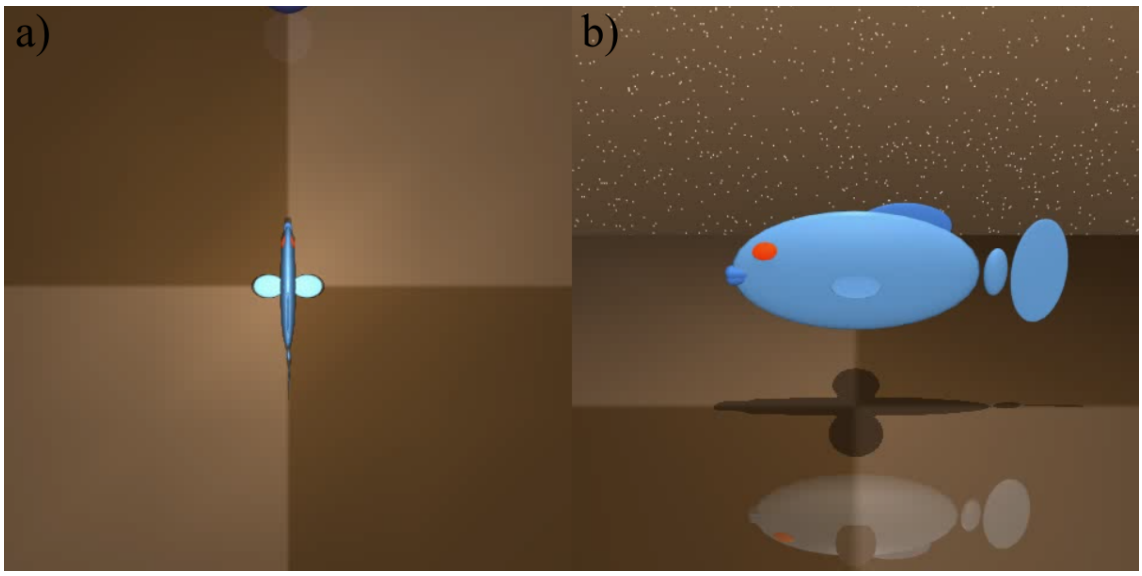

Figure 3.19 – Example image of modified fish environment from *dmcontrol* package [Tassa et al., 2018] using mujoco engine : a) top view (used for RL training) b) side-view.

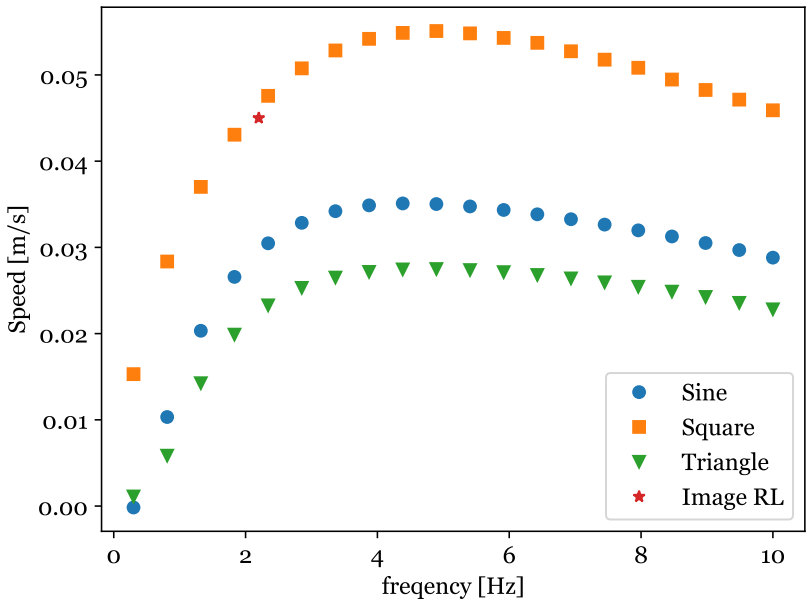

Figure 3.20 – Simulation results of the fish 1D swimming environment using mujoco engine. As it can be inferred the square wave forcing of the fin is the most powerful with the sinusoidal coming in front of the triangular forcing. "Image RL" corresponds to learning from visual input with DrQ-v2 that was able to learn the square wave forcing of the fish from visual data, but failed to optimize the frequency. This can be rationalized due to the low visibility of the fin from the top view.

2D direct numerical simulations : COMSOL

To transpose our results to a real swimming problem, we conducted simulations for the complete fluid-structure interaction in a 2D configuration (see Appendix E.1). In this numerical setup, the swimmer moves within a tank filled with liquid having the same density and viscosity as water. We consider the fish body to be viscoelastic, and we adjust its parameters to match the values of ω_0 and ξ obtained from the robotic fish experiments. The entire body measures $L = 10$ cm with an average thickness equal to $H = 1.3$ cm. To model muscular activity, we impose a spatiotemporal variation of the elastic body length at equilibrium. In particular, we set that the equilibrium of the strain component ϵ_{xx} varies parabolically as a function of the distance from the head X, and linearly from the midline Y : $\epsilon_{xx} \propto (X/L)^2(Y/H)a(t)$, where $a(t)$ varies temporally. With this functional form, the half body ($Y > 0$) extends its length while the other part ($Y < 0$) retracts, resulting in the swimmer bending to compensate for the inhomogeneous change in length across the body thickness. We have simulated the motion of this 2D active elastic beam embedded in water, driven by different functional forms of $a(t)$. In Figure 3.21, we present the cruising speed achieved with square, sine and triangle wave functions at various control frequencies (f). Remarkably, the square wave forcing consistently leads to the highest speed, regardless of the frequency, confirming the predictions from the RL algorithm and the model.

All the three forcing gaits exhibit a peak at a frequency close to 3.1 Hz. Additionally, we have implemented the swinging strategy control where we keep $a(t)$ constant in absolute value but change its sign as the vertical velocity of the tail (at $X = L$) reaches zero.

The swinging controller automatically selects a frequency that brings the swimmer to near maximum speed as depicted in Fig. 3.21. Therefore, our numerical simulations validate our interpretations regarding the mechanisms to achieve the highest swimming speed.

Discussion

We have given a thorough analysis of simulations in three different virtual environments for a robotic fish. All of them converge on the fact that the bang-bang control is optimal and there exists an optimal frequency for speed maximization. Moreover, sinusoidal forcing control is more performant than a triangular one and swinging strategy remains a robust model-free control for speed maximization. The optimization of a simulated speed from a visual data is feasible and requires more time than the optimization from a true state. With acquired knowledge, we now proceed with mean speed optimization of a real robotic fish from visual data.

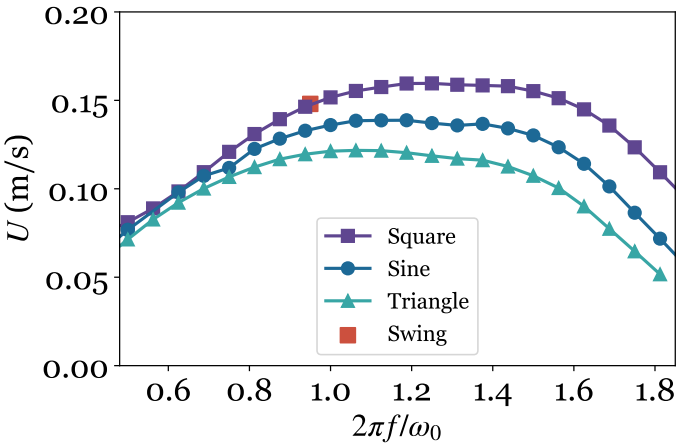

Figure 3.21 – Computation of the swimming speed using a full 2D Fluid-Structure-Interaction simulation. The blue symbols are obtained using a sinusoidal driving, the green symbols with a square wave forcing, and the red symbols are computed with a triangular function.

3.3.2 Experimental setup and methods

In our experiments, we used a 3D printed robotic fish rear body described in Section 3.2.1. Our initial design involved using a linear guide setup to propel the robotic fish. However, this approach faced several challenges, including mechanical imperfections and friction, which hindered the fish's intended forward movement. Additionally, tethering the fish to a power source via a wire further compromised its maneuverability. To address these challenges, we opted to control a robotic fish wirelessly and without a static source of power. This eliminates mechanical imperfections and wires that could impact the optimal swimming gait we aim to find. Communicating with underwater robotic systems is challenging, often necessitating the use of medium-range radio frequencies such as 433 MHz for effective data transmission [Katzschmann et al., 2018]. For the sake of simplifying our experimental setup, we opted for Wi-Fi protocol based communication, with all the electronics components of a robotic fish put inside the plastic tube ($Fisherbrand^{TM}$ 500ml with 53mm diameter) floating on the surface of the water. In order to mitigate the risk of harmonic pollution, we implemented separate power supplies for the servo-motor and the primary electronic board (Raspberry Pi). We used a block of four 1.5 V, 3000mAh rechargeable batteries to power the servo-motor, while a separate battery of 3.7 V, 1200 mAH powered the main card "Raspberry pi Zero WH" through the "PiJuice Zero" power module. This separation ensured that the power supply for the motor did not interfere with that of the main electronics board, thereby maintaining the integrity of our experimental setup.

An overhead web-camera (ODROID USB-CAM 720P) captured the robotic fish movements [see example image in Fig. 3.22a]. Although both IMUs [10] and overhead mounted cameras are useful sensors for the estimation of robot position, the IMUs can introduce biases, particularly when deployed over extended periods. Thus, we only use the camera to observe the robotic fish. Tracking objects within images generally falls under two broad categories : marker-based and marker-less methods, as outlined in [Mathis et al., 2020]. The first uses standard image processing techniques to infer position of markers on the object, while the second uses neural networks to determine the tracking position of an object. To simplify and accelerate the training process, we implemented the marker-based approach by affixing yellow and red markers to the buoyant cylinder. This approach eliminates the use of high-dimensional input images in the RL algorithms. One marker suffices to determine an object's position precisely ; however, to infer the orientation of a robot necessitates at least two markers. The objective of a robotic fish was to swim upstream, maximizing mean velocity over a one-second duration in alignment with the reservoir's orientation. The idea is that the robotic fish gradually learns to swim upstream. Once the episode terminates, the robot is brought back to the initial point with the water flow during the reset.

All the experiments were conducted within the water tunnel of Rolling Hills Research Corporation, detailed in Appendix D.1. The water flow was regulated to maintain a velocity of 60 mm/s in a direction opposite to that of the robotic fish. During training, the robotic fish swam upstream and the water flow of 60 mm/s ensured quick and effective reset of the environment. During the training phase, this specific water flow rate served a dual purpose. Firstly, it provided a consistent opposing force for the robotic fish, effectively simulating the challenges of swimming upstream. The increased drag force ensured that only effective control policies were able to propulse the robotic fish forward. Secondly, the 60 mm/s flow rate facilitated a rapid and efficient RL environment reset. The learning episode is ended by halting the fish, and fish going backwards with the flow of the water tank. To prevent the fish from becoming immobilized during the reset due to fric-

10. IMU : inertial measurement unit

tion along the reservoir's boundaries, fish undulates its tail slightly every three seconds. Once the camera detects a certain position threshold along the x axis is overcome, the fish restarts the new episode again. Small 3D-printed blue piece of a "triangular" shape is mounted at the beginning of the track, so that the robotic fish starts at about the same position relative to the reservoir.

The agent-environment interaction during learning occurred in episodic time segments of 128 time steps with a sampling time of 90 ms. The episode time of 128 time steps ensures that the robotic fish always stays in the overhead camera vision view and a sampling time of 90 ms guarantees consistent Wi-Fi data exchanges without timing irregularities. The episode stops upon reaching the maximum time step count. At every time step, the agent receives the reward proportional to mean displacement speed projected on x axis, averaged on the $1s$ time interval. To infer the first and second derivative of the longitudinal and angular positions of the system, 4 consecutive position values of two markers together with the actuation angle ϕ_c were part of the observation space of the RL environment, just like in the training with raw images. The resulting observation at time step i is $[\phi_{ci}, cx_i, cy_i, px_i, py_i]$x4 (concatenated with 3 previous time steps), where (cx_i, cy_i) are the coordinates of one marker in 2D space at time step i and (px_i, py_i) are the coordinates of the second point.

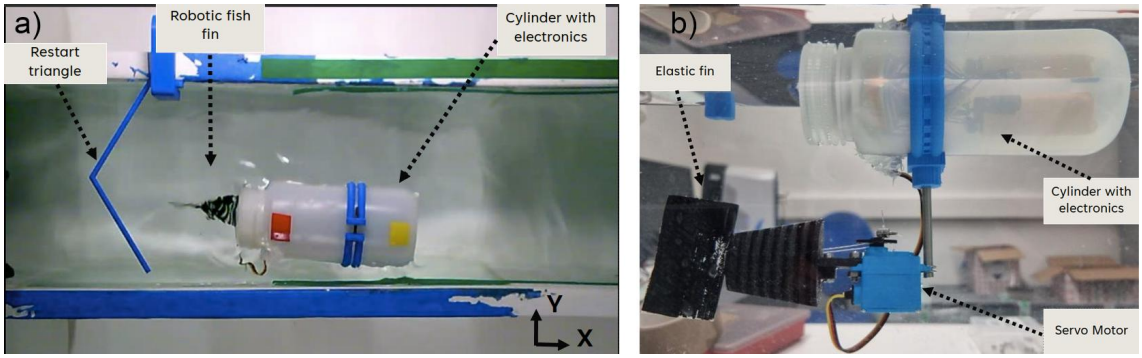

Figure 3.22 – Experimental setup for a robotic fish swimming in a water tunnel. a) The use of two points permits to define the position of the swimmer, as well as its body angle orientation with respect to the flowing water direction. b) Side view of the setup : the servomotor actuates the elastic fin of a robot fish via the fishing wires, which in turn propels the entire system against the direction of the flow. All the electronics with batteries are located in the plastic tube floating on the surface of the water.

3.3.3 Experimental results : speed maximization

We employed PPO [Schulman et al., 2017] algorithm with discrete actions to optimize the swimming speed. The algorithm used four consecutive values of $[cx_1, cy_1, px_2, py_2, \phi_c]$ as an observation, a reward of mean speed $\dot{x}$ along the x-axis over $1s$ and action of $\phi_c \in \{-\Phi, 0, \Phi\}$, where $\Phi = 40°$ is a maximum servo-motor actuation angle. The found policy after 120000 time steps was square wave [see Fig. 3.23] with mean speed increasing rapidly at the beginning and saturating towards the constant value towards the end of the training.

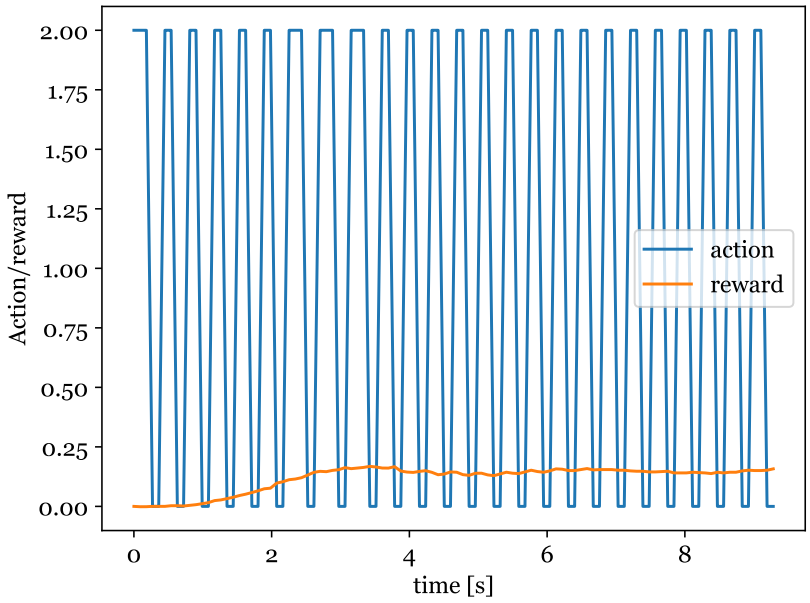

Figure 3.23 – Experimental results in inference for a robotic fish swimming in a water tunnel after 120000 time steps with sampling time of 90ms. The base frequency is 4 sampling times, which translates to the actuation frequency of f=2.74 Hz. The actions are discretized suggesting that 0 corresponds to one extremity and 2 to another.

Discussions

We have successfully demonstrated that the mean speed of a robotic fish can be optimized using the visual data and predetermined markers on a fish body. The main purpose of putting the markers is to indicate the useful features from high-dimensional images without training a feature extractor. Having already processed information from images should boost the training speed. The downside of the approach is that two markers may not be enough to capture the whole complexity of the robotic fish movement in the water tank.

We applied three preset control policies to propulse the robotic fish : square wave, sinusoidal and triangular. In the conducted experiments, sinusoidal and triangular waveforms for actuation are not represented here. This omission is intentional, as these waveforms were found to be less effective than square wave forcing for enabling the robotic fish to swim against the direction of water flow. Specifically, during trials employing triangular waveforms at various frequencies, the fish was almost always unable to make headway against the current for frequencies $f < 2Hz$.

We now evaluate the efficacy of three distinct approaches in Fig. 3.24 : RL optimization of the mean speed using camera, the square wave forcing policy and the thrust force optimization of a stationary fish via RL using a force sensor. We found an optimal policy to be 3 Hz which closely matches the frequency obtained via RL optimization. Additionally, when we conducted thrust force optimization experiments using the same robotic fish in a static condition, we observed that the optimal frequencies ($\approx$ 2 and 3 Hz) were closely aligned. This suggests a good level of robustness in the RL optimizations of thrust and speed, further validating the effectiveness of bang-bang control for the propulsion of a robotic fish.

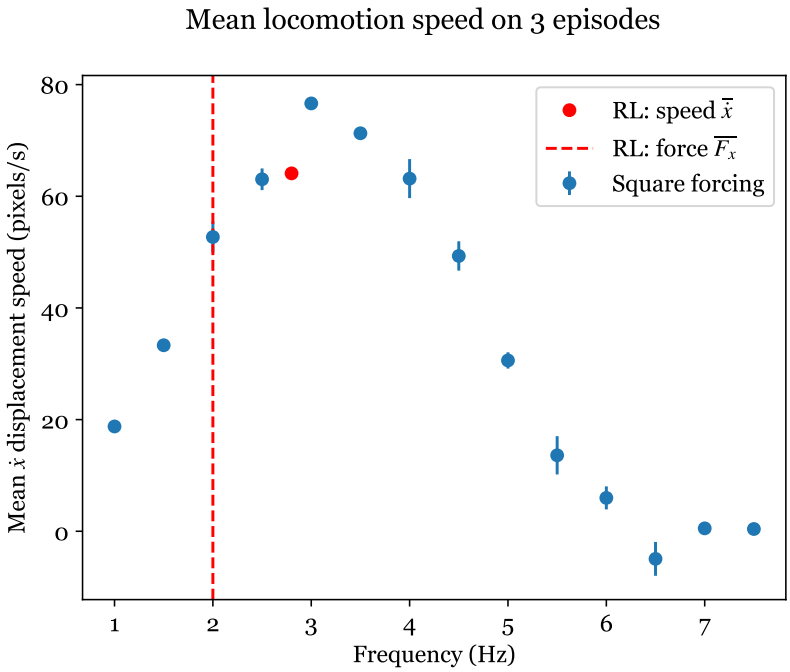

Figure 3.24 – Comparison of different approaches for finding an optimal swimming gait. Blue symbols corresponds to forcing with square wave policy, the discontinuous red line to the optimal frequency found of thrust optimization and red symbol to the speed maximization via RL.

While maximizing the thrust force is equivalent to maximizing the speed from the physical perspective, this simplified perspective neglects a nuanced physical phenomena that can come into play in experiments. Primarily, the undulatory motion of a fish's tail generates waves that propagate through the water, altering flow conditions and potentially influencing the fish's movement [Liao, 2007]. The wave propagation (referred as sloshing) in the water tunnel is highly dependent on the geometry and the size of a fish and the tank. We conducted experiments with robotic fish of different sizes and concluded that in our experimental setup, sloshing influence was not significant for the robotic fish we used before [see Fig. 3.1]. However, when we tested a larger robotic fish [Berg et al., 2021], we observed a noticeable sloshing effect at the frequencies higher than f_c = 1.8 Hz of oscillations severely interfering with the thrust force measurements.

Sometimes positional reward can be hard to define. In many robotics application, it is hard to define the objective quantitatively yet easy to visualize with an image. We now proceed with a visual servoing task in a virtual environment for fish swimming.

3.3.4 Visual servoing applied on a simulated robotic fish

One of the common applications in robotics is to manipulate the object using camera readings. Visual servoing [Chaumette and Hutchinson, 2006] is a well-established technique in control theory that uses cameras as a position sensor. One of the classical approaches of visual-servoing is an image-based visual servoing [Chaumette and Hutchinson, 2006] (IBVS), where the control is happening directly in the image space with the minimization of an error between the current and desired image plane features. Another visual-control approach is position-based visual servoing [Chaumette and Hutchinson, 2007] (PBVS), which requires the calculation of the estimated pose of the camera as well as the object of interest in 3D. The controller uses traditional methods and minimizes the Cartesian distance of the target and current positions. Tasks like 3D localization typically fit into PBVS. Conversely, no pose estimation is required in IBVS. The relation between image space and Cartesian space is given in the "interaction matrix". It is an estimation of the relationship between camera motion and image-plane motion. The schema block of two methods is illustrated in Fig. 3.25. Here, we focus on visual servoing method which uses extracted useful features from high-dimensional images for the precise positioning in the image space with Auto-Encoders [Felton et al., 2022]. In the context of a robotic fish, visual servoing can be utilized to control the fish's motion so that it reaches and maintains a desired position or orientation relative to a visual target.

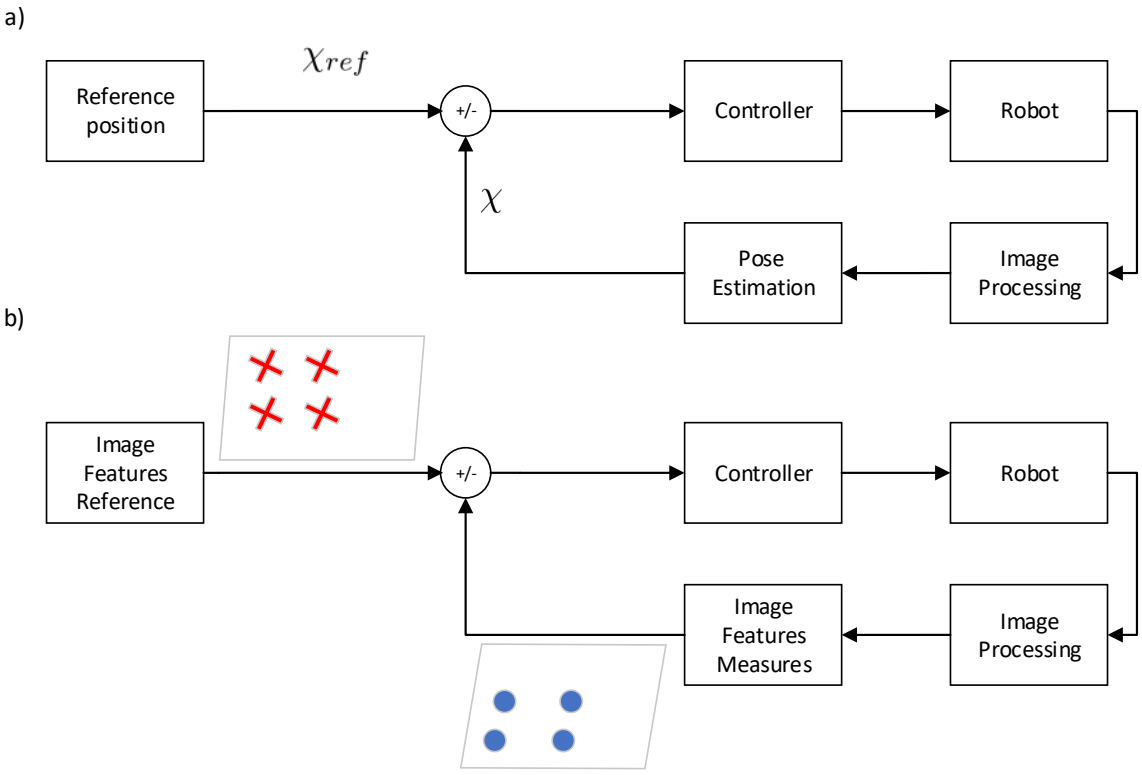

Figure 3.25 – control block diagrams comparison : a) PBVS. Control is happening in cartesian space. χ_{ref} and χ refer to to target and current positions. b) IBVS. Control is happening in image feature space.

In many cases, it is possible to determine useful features of an object from visual data using standard image processing techniques, for instance using colored markers [DeGol et al., 2017] or optical flow [Mahony et al., 2007]. Image processing tools, for tasks like segmentation, feature extraction or object tracking often require manual parameter-tuning or domain-specific expertise. Recently, CNNs opened new venues of extraction of useful features from images [Krizhevsky et al., 2012, Muhammad et al., 2018, Liu et al., 2022c]. Artificial neural networks, particularly CNNs, demonstrated extraordinary capabilities to extract the useful features from images [Geirhos et al., 2017]. In some cases, images have to be accurately labeled by the human to produce the useful features with CNNs ; pretrained CNNs on large-scale annotated datasets such as Imagenet [Krizhevsky et al., 2017, Beyer et al., 2020] can be used as a feature extraction tool on the downstream tasks such as reduced space representation and visual servoing [Lee et al., 2017]. Very often, pre-trained CNNs are later fine-tuned on the downstream tasks for the better performance.

However, there exist unsupervised [Kingma and Ba, 2014], contrastive and self-supervised algorithms [Caron et al., 2020], that permit to extract the useful features without human annotation. Those pretrained neural nets can then be used for downstream tasks. One of such unsupervised deep learning algorithms, Variational Auto-Encoder [Kingma and Ba, 2014, Kingma and Welling, 2014b] can encode input images into the low-dimensional space, and this low-dimensional space can be coupled with deep RL to address visual servoing tasks [Nair et al., 2018, Pong et al., 2019]. Contrary to CNN, VAE adds the disentanglement information on images which can be used in the visual servoing context.

Autoencoder Autoencoders [Kramer, 1991] is a type of ANNs that is employed to learn efficient data encoding in an unsupervised manner. It consists of a double model : encoder and decoder networks that are trained simultaneously. An encoder network q_ϕ takes image $\mathbf{X}$ as an input and outputs a latent vector $\mathbf{Z}$. A decoder net p_θ, on the other hand, takes a low-dimensional latent vector $\mathbf{Z}$ as an input and reconstructs an original image [see Fig. 3.26]. Autoencoder parameters are optimized via either MSE [11] [see Eq. (2.56)] loss of original and reconstructed images or log-probability loss [see Eq. (3.20)]. Basically, an autoencoder is trained to replicate its input to its output. Because of the low-dimensionality of the latent space, autoencoders can not copy perfectly its input to the output, but instead they learn the important parts of the high-dimensional input that can be used in the downstream tasks. Some of the applications of autoencoders include image compression, dimensionality reduction and denoising. Autoencoders are trained without imposed structure on the latent space, which renders them not suitable in general for exploiting the latent space. Variational Auto-Encoders (VAE) described in the next paragraph is capable of encoding and decoding a high-dimensional input via more structured latent space.

Variational Auto-Encoder (VAE) VAE [Kingma and Welling, 2014a] builds on top of an Autoencoder with a probabilistic latent space. Instead of having some discrete values in the latent space like it was with AE, the latent space of VAE is composed of distributions, typically gaussians [see Fig. 3.27]. This property makes a VAE a generative model that enables to generate new high-dimensional data by sampling the latent space distributions and then reconstructing the output through the decoder. The encoder produces a probability distribution over the latent space and decoder reconstructs the high-dimensional input from the sample of probabilistic latent space distribution. The probabilistic latent space is achieved via a constraint on the latent space during

11. Mean Squared Error

training. This constraint makes the variables in the latent space to follow the imposed distribution, such as normal distribution.

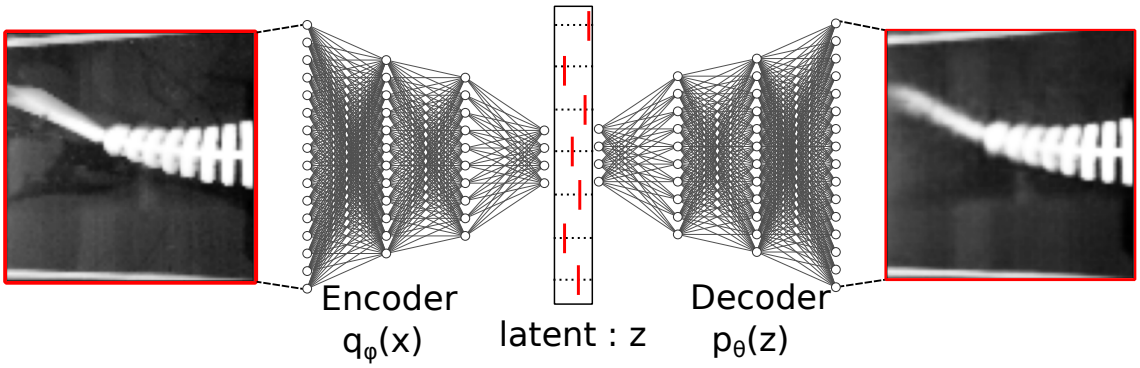

Figure 3.26 – Example of AE reconstruction of the real image of the fin.

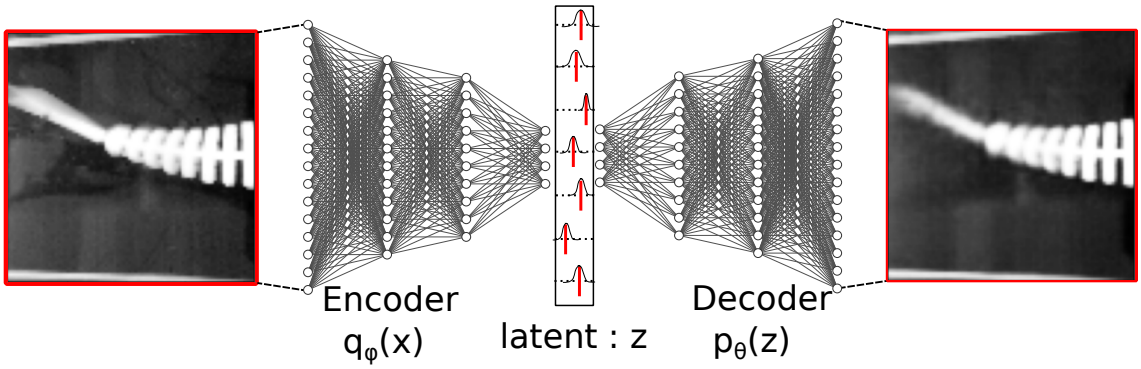

Figure 3.27 – Example of VAE reconstruction of the real image of the fin.

The comparision of Autoencoders and VAE is illustrated in Fig. 3.28. In practice encoder of VAE outputs the mean and standard deviation of the distribution (μ, σ) instead of a latent vector in Autoencoders. Some value z is sampled from this distribution for the reconstruction [see Fig. 3.28]. The distribution sample is defined as $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma}$. This is referred as "reparametrisation trick", which allows for the backpropagation of the loss through the encoder-decoder networks to optimize both mean μ and standard deviation of a distribution σ.

To understand how VAEs constrain the latent space, one must understand the concept of Kullback–Leibler divergence or the relative entropy. For two distinct one-dimensional distributions $p(x)$ and $q(x)$ of a variable x, Kullback–Leibler (KL) divergence $D_{\mathrm{KL}}(p(x)\|q(x))$ is a non-symmetrical positive statistical measure indicating how distant two distributions are from each other. It is defined as :

$$D_{\mathrm{KL}}(p(x)\|q(x)) = \int_{-\infty}^{\infty} p(x) \ln \frac{p(x)}{q(x)} dx. \tag{3.19}$$

For an illustrative example, we take two sample distributions in Fig. 3.29a. The $D_{\mathrm{KL}}(p(x)\|q(x))$ is the sum of the shaded area in Fig. 3.29b. The positiveness of KL-divergence make it suitable to be used in the loss function while training VAE neural networks (encoder and decoder). The

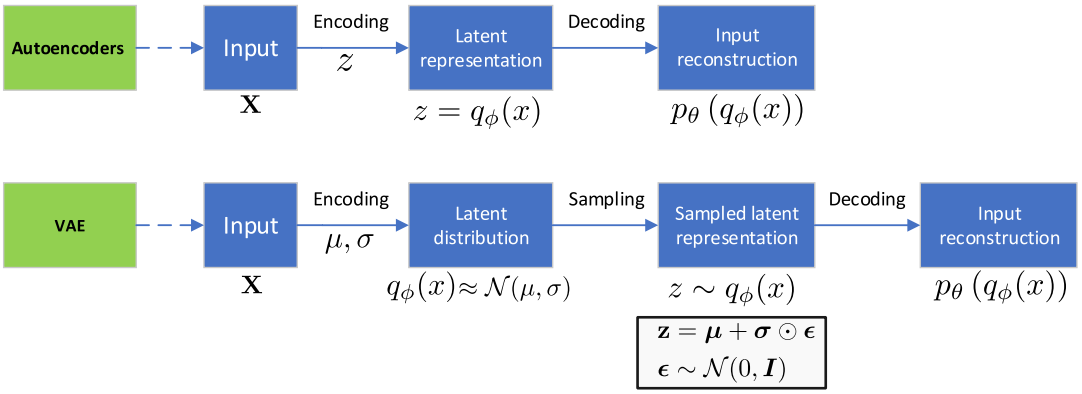

Figure 3.28 – Schema-block of functioning for Autoencoder and VAE.

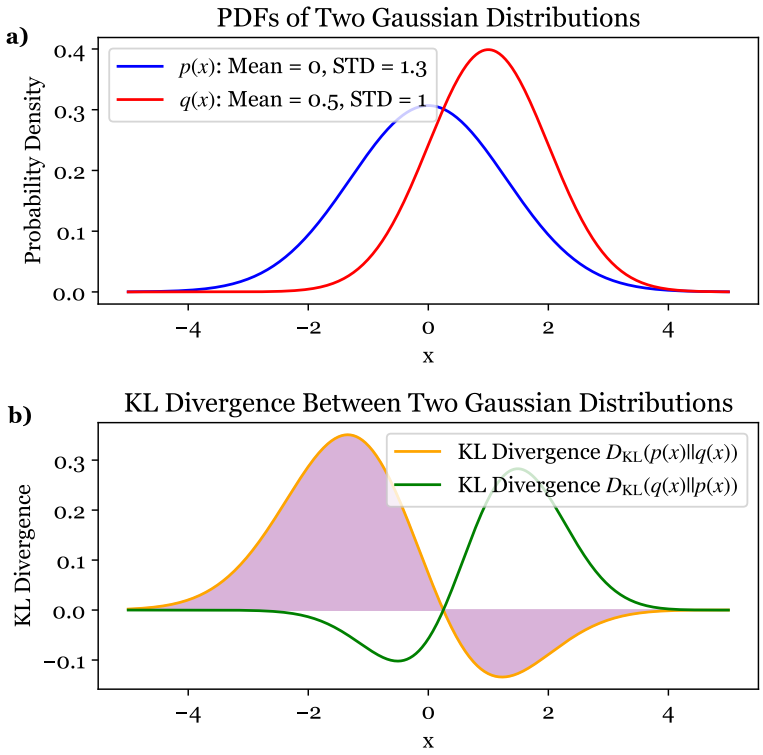

Figure 3.29 – a) Two probability density functions (PDF) with different means and standard distributions. b) KL divergences of two distributions. The shaded area corresponds to $D_{\mathrm{KL}}(p(x)\|q(x))$.

general form of loss functions in the training of AEs and VAEs takes the following form :

$$\mathcal{L}^{AE}(x, \theta, \phi) = -\log\left(p_\theta\left(q_\phi(x)\right)\right). \quad \text{or} \quad \left[x - p_\theta\left(q_\phi(x)\right)\right]^2 \tag{3.20}$$

$$\mathcal{L}^{VAE}(x, \theta, \phi) = \mathbb{E}_{q_\phi(z|x)}\left[-\log\left(p_\theta\left(q_\phi(x)\right)\right)\right] + KL\left(q_\phi(z|x)\|p(z)\right). \tag{3.21}$$

The latent space of VAEs is constrained via the KL-penalty $D_{\mathrm{KL}}(q_\phi(z|x)\|p(z))$ in the loss function [see Eq. (3.21)]. At every training step, the algorithm updates the weights of a neural

net so that the encoded distribution $q_\phi(z|x)$ is close to $p(z)$, a normal Gaussian in our case. This KL-penalty encourages the latent space distribution to follow the gaussian form with mean $\mu = 0$ and variance $\sigma^2 = 1$. In some cases, Bernouli distribution is used for the KL-penalty to represent the binary input data [Loaiza-Ganem and Cunningham, 2019].

As noted before, one of the powerful applications of AEs and VAEs for control is a feature extraction from high-dimensional images, a process known as "State Representation Learning" [Lesort et al., 2018, Raffin et al., 2019b]. This can be useful to decompose end-to-end learning control from visual data into two different steps : learning the compressed state representation with reconstruction and learning the control separately or jointly. Learning the control from raw images may be the hard task and it is often more intuitive to separate perception from control by decomposing ent-to-end control learning from visual data into 2 easier sub-problems. The only disadvantage of this approach is that AE/VAE reconstructs the image which is not necessary for learning the successful control policy. However, it is easier to tune the hyperparameters and interpret the problems after a successful training of the feature extractor. While the use of this method may result in increased computational overhead compared to reconstruction-free techniques due to the training of the decoder, it can yield better sample-efficiency and it is easier to interpret.

Another useful application of VAE for control is comparison of the proximity of images. There are many variants of VAEs, such as $\beta-$VAE [Higgins et al., 2017] introducing the β hyperparameter to the KL-penalty, which introduces the trade-off for image reconstruction and constraining the latent space. The latent space of VAE is compact and can be used to infer the information about the original image. Two images different in the high-dimensional space will have two distant latent encodings in the euclidean space. This property of VAEs can be exploited in the context of visual servoing when the goal of the controller is to reach the target image [Felton et al., 2022]. We now begin by formulating the problem of visual servoing and then move on to define the task in a simulation environment, followed by presenting the obtained results.

Learning to achieve a goal image position

In the context of MDP, let the high-dimensional image state be $\mathbf{X}_t$, the control $\mathbf{U}_t$ be defined by the policy $\pi : \{\mathbf{X}_t \rightarrow \mathbf{U}_t\}$, the trajectory of T consecutive states defined by τ and feature extractor mapping defined by $g : \{\mathbf{X}_t \rightarrow \mathbf{Z}_t\}$. Let at time step t, $\mathbf{Z}_t$ be a latent encoding of the camera's observation $\mathbf{X}_t$ and let $\mathbf{Z}_*$ be some given goal latent state. We define visual servoing as the problem of choosing a sequence of controls $\mathbf{U}_{1...T}$ over the trajectory τ with T discrete time steps as to minimize the latent distance error between the goal encoding $\mathbf{Z}*$ and the the final encoding of the system state $\mathbf{Z}_T$, i.e $\|\mathbf{Z}_* - \mathbf{Z}_T\|$. The advantage of this type of unsupervised reward is that it does not require human knowledge and reward shaping. The optimized policy chooses the sequence of control steps $\pi(\mathbf{X})$ to reach a goal image and has the following objective :

$$\theta = \underset{\mathbf{X_T} \sim \pi_\theta(\tau)}{\arg\min} \|\mathbf{Z}_* - g(\mathbf{X}_T)\|^2 ,$$

where θ are the parameters to be optimized. Thus, during whole episode the agent will not receive any learning signal except at the final time step of the episode. This is one of the formulations of the problem, accounting only for the final goal image. This formulation is challenging and has a sparse learning signal for RL algorithms that are emitted only at the end of an episode of duration T. This formulation addresses the goal-conditioned RL [Liu et al., 2022a, Andrychowicz et al., 2017] and it incorporates additional computational/memory constraints that are detailed in [Nair

et al., 2018]. To make the reward more informative, we can sample the informative reward at each time step and we proceed with the mean formulation on the episode duration :

$$\theta = \underset{\mathbf{X_t} \sim \pi_\theta(\tau)}{\arg\min} \frac{1}{T} \sum_{t=0}^{T-1} \|\mathbf{Z}_* - g(\mathbf{X}_t)\|^2$$

Simulation results

We now construct a simple visual simulation of a robotic fish with the dynamics defined in Eqs. (3.17) and (3.18) that swims in one direction and is constrained in the others. The fish starts at the initial position and at every time step the agent is penalized for being far away from the target image. The penalty is defined as an euclidean distance in the latent space between the goal and the current latent encodings defined by VAEs. Thus, the agent's objective is to make fish reach the objective goal image from the initial image in the shortest time possible [see Fig. 3.30]. We reset the environment when the agent reaches the goal image or maximum time steps. We define that the agent reaches the goal image when $\|\mathbf{Z}_* - g(\mathbf{X}_t)\|^2 < \epsilon$, where ϵ is a hyperparameter chosen manually.

First, we construct a dataset of simulated images with different parameters (x, α), then we train VAE for several epochs on this diverse simulated data. Then the encoder part of VAE is used together with PPO on a visual servoing task. The observation of PPO is a latent encoding via encoder of the current image of the simulator (simulator processed image : 84x84x1 $\rightarrow$ latent mean value : 8x1). The reward is the latent euclidean distance between the current and the goal images.

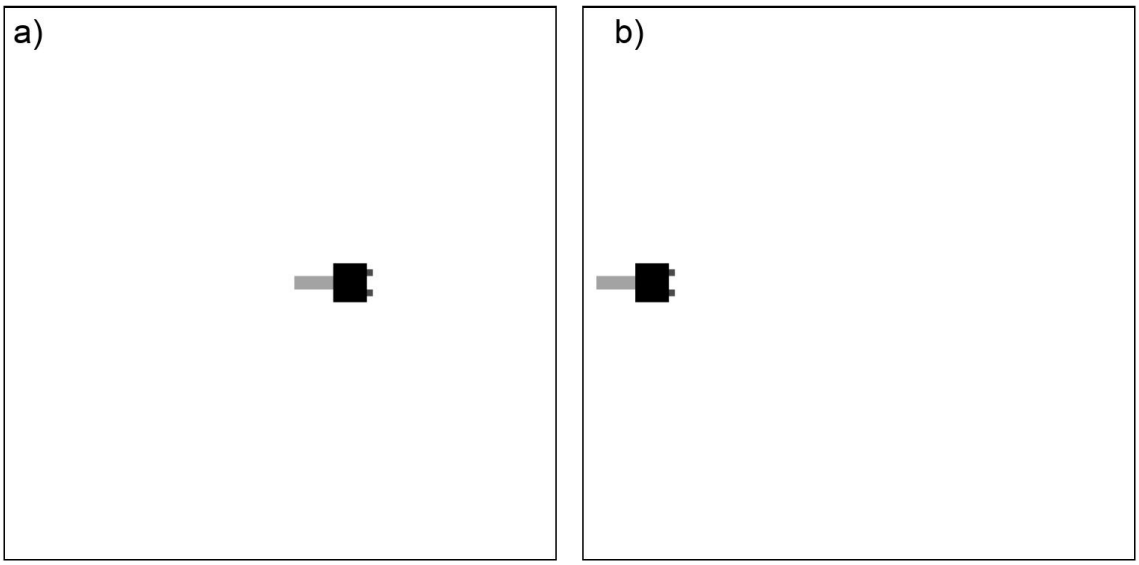

Figure 3.30 – Processed images for the visual-servoing task in simulation : a) the goal observation of the task. b) the initial observation of the system at the beginning of an episode.

We present the RL simulation results with the custom $\beta-$VAE [see the architecture in Table B.5] in Fig. 3.31 and the learning statistics shown in Fig. 3.32. To further validate the results, we employ square wave, sinusoidal and triangular predefined functions to control a robotic fish with different frequencies in Fig. 3.31a. From Fig. 3.31a, it can be inferred that PPO is

able to find the optimal swimming gait and frequency. In this figure, there are some outlier values among the square wave forcing; this is due to large distances in the latent space between some images that VAE is not confident at encoding. One of the further ways to smoothen the negative spikes is to clip the large negative penalties.

The optimal swimming gait is square wave and the optimal frequency obtained with this visual servoing environment is the same as the one obtained when the objective was to maximize $\dot{x}$ from the true state $[x, \dot{x}, \alpha, \dot{\alpha}]$ seen in Section 3.3.1, because of the same underlying dynamics. Thus, this visual-servoing approach comprising deep RL coupled with VAE can be seen as a viable approach to maximize the swimming speed when only raw visual data is available. Training the algorithm for 400000 time steps is too long for the experimental setup. Further work includes implementing more sample efficient algorithms before the application in practice.

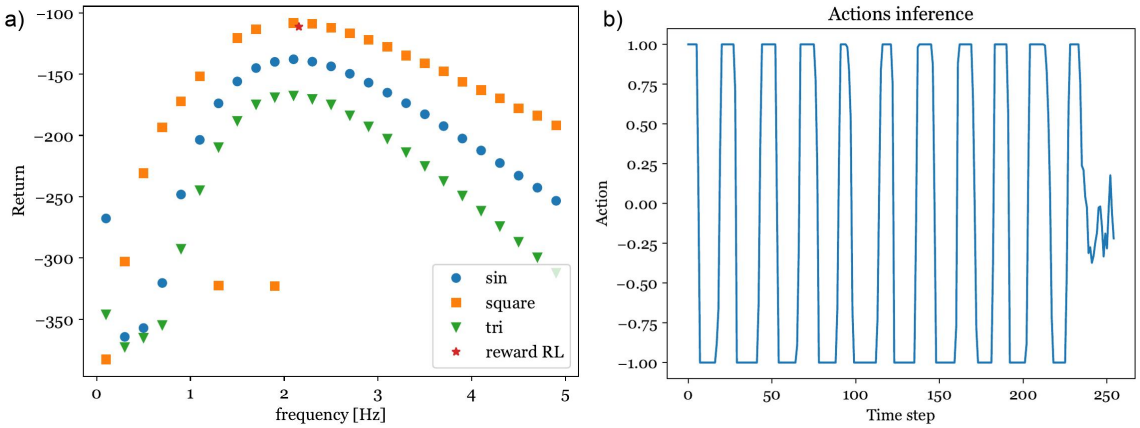

Figure 3.31 – RL simulation results of a fish visual servoing environment. a) Influence of the frequency on the episodic return by some periodic functions : square wave (orange squares), sinusoidal (blue circles) and triangular (green triangles) waves. b) Control actions during one episode from the best learnt policy in inference.

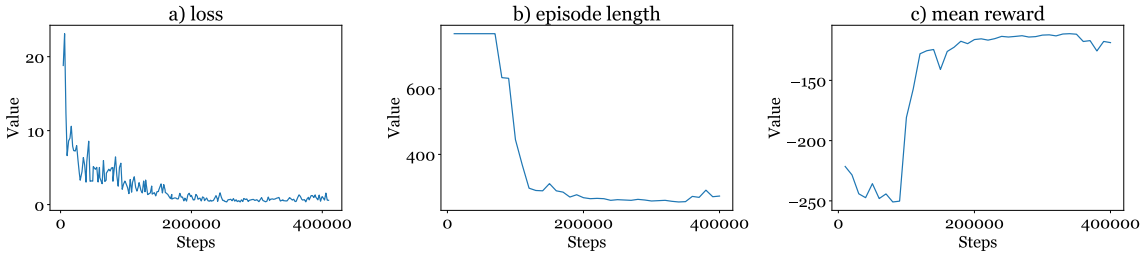

Figure 3.32 – Learning statistics of visual servoing environment with PPO. a) The loss being minimized is steadily heading zero until the convergence at about 300000 time step. b) average length of the episode at inference evaluations, stays the same at the beginning for untrained agent. Then at around 100000 time step, the agent learns to achieve the goal and the episode length decreases drastically. c) mean inference return, which has a drastic increase at 100000 time step that can be rationalized by the previous plot.

CHAPTER 4

Conclusion and Perspectives

4.1 Conclusion

This thesis shed new insights on the control of robotic systems via deep Reinforcement Learning methods, particularly thrust and speed optimization of a robotic fish. The primary objective of the thesis was understanding the fish locomotion and designing optimized ways to actuate robotic fish for thrust and speed maximization. Before applying deep RL algorithms on a robotic fish setup, we performed an extensive study of these data-driven algorithms on a traditional benchmark problem of a control theory : cart-pole *i.e.* inverted pendulum.

Initially, in Chapter 2, we presented classical control algorithms : PID, LQR and Lyapunov-based controllers, highlighting their advantages and downsides. Then, we applied these model-based control techniques on a real cart-pole system. Lyapunov-theory based controller was able to swing-up and LQR to maintain the cart-pole in its unstable equilibrium.

Subsequently, we described the model-free RL framework with three classical families of algorithms. Different deep RL algorithms were then presented with the application on an inverted pendulum and thorough analysis of the algorithm performance. Specifically, we studied the application of Q-learning and DQN in detail on a real experimental setup. It has been demonstrated that DQN was capable of swinging up and stabilizing the pendulum in its unstable equilibrium in a reasonable time. The impact of various physical parameters on the robustness of the DQN algorithm's learning process was investigated. We also demonstrated the role of the measurement and actuation noise on the learning capabilities. Finally, we compared the performance of different discrete and continuous deep RL algorithms on the swing-up and stabilization of a simulated inverted pendulum.

In Chapter 3, we studied the application of deep RL algorithms that could maximize the trust and the speed of a robotic fish. First, we presented a physical model of a robotic fish as a dual dynamics formulation : dynamics of a soft body of a robotic fish and servo motor internal dynamics. We demonstrated that the found control strategy was optimal to maximize the thrust force. We demonstrated the robustness of obtained results with different learning sessions using the camera and force sensors for observations. It was revealed that both learning in simulation and the analytical solution of the swimming model yielded results comparable to RL learning in experiments. The resulting bang-bang control was explained and rationalized through the Pontryagin maximum principle. Finally, we presented the swinging strategy that uses the angular speed of the robotic fish tail to propel the robotic fish at high efficiency. We have successfully demonstrated that the bang-bang control is optimal and there is an optimal frequency of the servo motor actuation. This

is explained via the elasticity of the fin ; moreover, the optimal forcing frequency is closely related to the natural frequency of the robotic fish fin.

In the second half of Chapter 3, we studied robotic fish speed optimization problems. Our previous findings regarding the optimality of bang-bang control were validated in simulation environments. We then demonstrated a learning protocol for maximizing speed in experiments using visual data from a top-mounted camera. It was found that the PPO algorithm was capable of identifying an optimal policy in experiments, and square wave forcing confirmed the results. Finally, we compared the results to the thrust force optimization and discussed the advantages and inconveniences of this optimization approach.

4.2 Perspectives for a robotic fish swimming

Our research has primarily concentrated on maximization of thrust and speed of a solitary aquatic swimmer. Further work encompasses the developments in mechatronics for an autonomous fish and enhancements of control via advanced deep RL algorithms.

In Fig. 3.22, we proposed a setup for swimming up the flow and control via external camera mounted at the top of the swimming reservoir. We also added IMU [1] inside the tube for further measurements of orientation and displacement of fish moving. Displacement determination requires double integration of noisy accelerometer values that also drifts slightly with time and temperature. IMU accelerometer measurements can be coupled with camera-based position tracking in sensor fusion processes to improve tracking robustness. The use of IMU also permits to determine the orientation of the fish in 3D. A sample result of an orientation determination using IMU while employing harmonic control is visualized in Fig. 4.1. We can observe that the roll and pitch angles remain approximately constant while the yaw angle oscillates slightly with 1Hz, the frequency of harmonic control. IMU will be particularly useful in the future, when an autonomous fish will be designed to navigate in 3D.

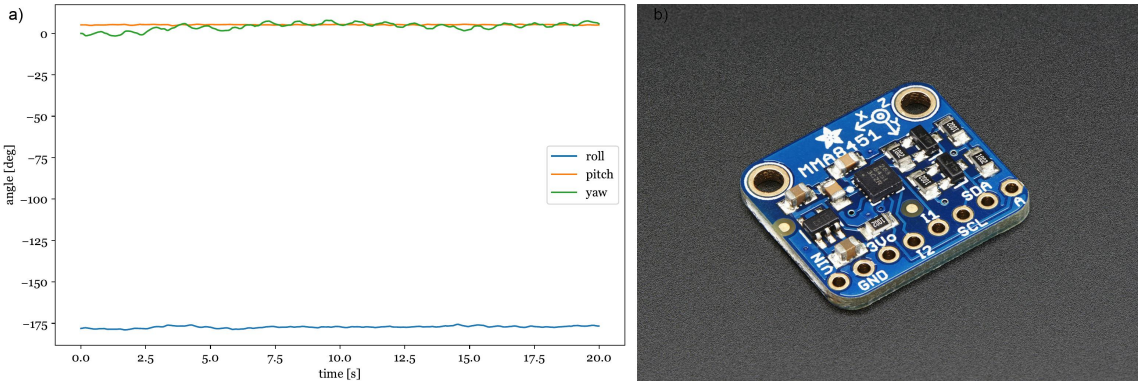

Figure 4.1 – Visualization of an IMU application. a) orientation determination of a robotic fish swimming in a straight line with sinusoidal forcing of 1 Hz. We used Madgwick algorithm [Madgwick et al., 2011] to estimate roll, pitch and yaw angles using IMU data. b) image of an IMU, taken from `www.adafruit.com` website.

1. Inertial Measurement Unit

For further development of a robotic fish, it is necessary to design a system for altitude control. The most intuitive method of regulating the altitude in a robotic fish is via pumping water in/out of the robotic fish, like the researchers did in [Katzschmann et al., 2018]. Another avenue for exploration is the possibility of achieving complete buoyancy and control of the altitude via pectoral fins with drag force. The adjustment of the robotic fish's orientation angle can be achieved by rotating its pectoral fins, enabling it to regulate the force by which the water pressures the robot up or down. By altering this angle, one could effectively modify the angle of the drag force exerted on the fish, thereby offering a mechanism for altitude regulation.

Attitude and depth regulation systems require a bigger robotic fish prototypes to house the necessary components. In our work, we explored a larger-scale robotic fish prototype, inspired by the design described in [Berg et al., 2021]. This model features a DC motor and a silicone-covered elastic tail for underwater propulsion, as seen in Fig. 4.2a. A significant limitation of this design is its dependency on a tethered connection for power, additional power drive electronics and external power source to drive the robot. We then experimented with a waterproof servo motor configuration instead of DC motor without silicone cover in Fig. 4.2b. For of the second configuration, we tested the thrust optimization for different harmonic control policies [done as before, see Fig. 3.5] and the results followed a similar trend with distinct peak frequency as the results obtained with a smaller robotic fish used before [configuration in Fig. 3.1].

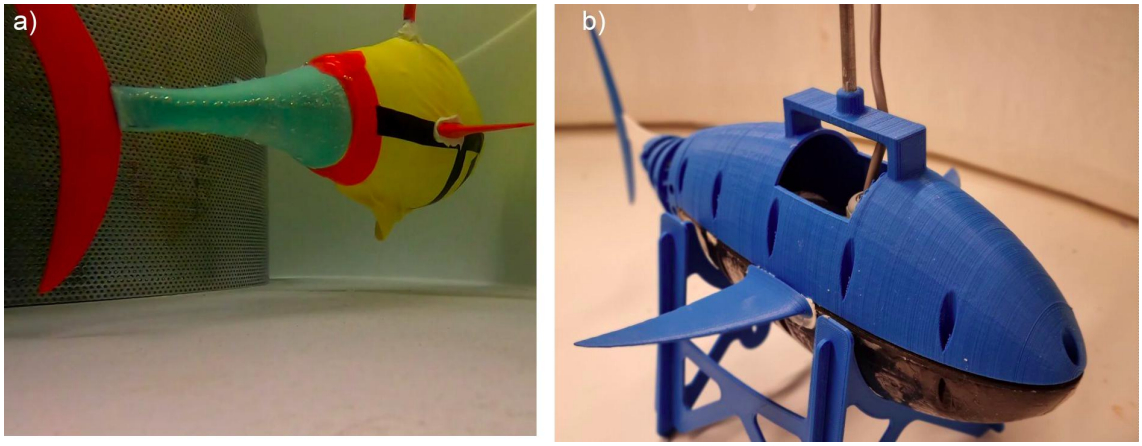

Figure 4.2 – Different tested variations of robotic fish conception, inspired from [Berg et al., 2021].

In the previous chapter, we demonstrated that RL remains a powerful method to optimize the thrust or speed. However successful RL application in practice requires many hours of real world robotic interaction. Like it was mentioned in chapter 2, some RL methods such as Q-learning needs several millions of interaction steps to maximize the reward which becomes infeasible in practice. Some machine learning techniques may improve sample efficiency. One of the potential ways is model-based RL [described in Appendix B.2] which premises improved sample efficiency sacrificing computational efficiency. Another useful technique is imitation learning or offline RL [see Appendix B.2 for more details]. The base idea is to use the knowledge of a potential good control to generate the trajectories that will serve to accelerate the training in a supervised way. We can use imitation learning on several episodes of square wave control to pre-initialize the actor model towards the bang-bang policy with certain frequency. One of the instances of imitation

learning, named "behavior cloning" has already been applied in simulation and yielded faster convergence.

Robotic fish can easily blend in the swarm of robotic fish. This is useful for marine biologists to observe closely marine life or ocean surface without disturbing the underwater animal realm [Fig. 4.3]. Many fish swim in swarm formations that enable them to harness energy efficiency by swimming in synchronized patterns [Hemelrijk et al., 2015]. Robotic fish in itself are more efficient than propeller-based UAVs. Forming a robotic fish swarm may further enhance their efficiency and multi-agent RL [Buşoniu et al., 2010] may be the key to efficiently control complex robotic fish swarm formations in chaotic environments. Deep RL can potentially unlock efficient strategies to propel a swarm of robotic fish due to the chaotic nature of the system.

Figure 4.3 – Robotic fish swims in the swarm of real fish, observing and recording the data. This image was created with the assistance of DALL·E 3.

Bibliography

OpenAI Gym. `https://https://gym.openai.com/`.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna : A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631.

Alexander, R. M. (2013). *Principles of animal locomotion*. Princeton University Press.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. *Adv. Neural Inf. Process. Syst.*, 2017-Decem(Nips) :5049–5059.

Angani, A., Lee, J. W., Talluri, T., Lee, J. Y., and Shin, K. J. (2020). Human and robotic fish interaction controlled using hand gesture image processing. *Sensors Mater.*, 32(10) :3479–3490.

Argentina, M. and Mahadevan, L. (2005). Fluid-flow-induced flutter of a flag. *Proceedings of the National Academy of Sciences*, 102(6) :1829–1834.

Åström, K. J. and Furuta, K. (2000). Swinging up a pendulum by energy control. *Automatica*, 36(2) :287–295.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv :1607.06450*.

Bainbridge, R. (1958). The speed of swimming of fish as related to size and to the frequency and amplitude of the tail beat. *Journal of experimental biology*, 35(1) :109–133.

Bakaráč, P., Kalúz, M., and Čirka, L. (2017). Design and development of a low-cost inverted pendulum for control education. In *21st International Conference on Process Control (PC)*, pages 398–403.

Baldi, S., Rosa, M. R., and Wang, Y. (2020). Model+ learning-based optimal control : an inverted pendulum study. In *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, pages 773–778. IEEE.

Barbera, G., Pi, L., and Deng, X. (2011). Attitude control for a pectoral fin actuated bio-inspired robotic fish. In *Proc. IEEE Int. Conf. Robot. Autom.*, pages 526–531.

Berg, S. C., Scharff, R. B., Rusák, Z., and Wu, J. (2021). Openfish : Biomimetic design of a soft robotic fish for high speed locomotion. *arXiv preprint arXiv :2108.12285*.

Berlinger, F., Gauci, M., and Nagpal, R. (2021). Implicit coordination for 3d underwater collective behaviors in a fish-inspired robot swarm. *Science Robotics*, 6(50) :eabd8668.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., J'ozefowicz, R., Gray, S., Olsson, C., Pachocki, J. W., Petrov, M., de Oliveira, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. *ArXiv*, abs/1912.06680.

Beyer, L., Hénaff, O. J., Kolesnikov, A., Zhai, X., and Oord, A. v. d. (2020). Are we done with imagenet ? *arXiv preprint arXiv :2006.07159*.

Boubaker, O. (2013). The inverted pendulum benchmark in nonlinear control theory : a survey. *International Journal of Advanced Robotic Systems*, 10(5) :233.

Boyer, F., Chablat, D., Lemoine, P., and Wenger, P. (2009). The eel-like robot. In *Proceedings of the ASME Design Engineering Technical Conference*, volume 7, pages 1–8.

Boyer, F., Porez, M., and Leroyer, A. (2010). Poincar'e-cosserat equations for lighthill three-dimensional dynamic model of a self propelled eel devoted to robotics. *Journal of Nonlinear Sciences*, 20(1) :47–79.

Boyer, F., Porez, M., Leroyer, A., and Visonneau, M. (2008). Fast dynamics of an eel-like robot—comparisons with navier–stokes simulations. *IEEE Transactions on Robotics*, 24(6) :1274–1288.

Brett, J. R. (1964). The respiratory metabolism and swimming performance of young sockeye salmon. *Journal of the Fisheries Board of Canada*, 21(5) :1183–1226.

Brett, J. R. (1972). The metabolic demand for oxygen in fish, particularly salmonids, and a comparison with other vertebrates. *Respiration Physiology*, 14(1) :151–170.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv :1606.01540*.

Bruzzone, G., Ferretti, R., and Odetti, A. (2021). Unmanned marine vehicles. *IEEE Robotics and Automation Magazine*, 28(2) :105–117.

Buşoniu, L., Babuška, R., and De Schutter, B. (2010). Multi-agent reinforcement learning : An overview. *Innovations in multi-agent systems and applications-1*, pages 183–221.

Buşoniu, L., de Bruin, T., Tolić, D., Kober, J., and Palunko, I. (2018). Reinforcement learning for control : Performance, stability, and deep approximators. *Annual Reviews in Control*, 46 :8–28.

Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. (2020). Unsupervised learning of visual features by contrasting cluster assignments. *Adv. Neural Inf. Process. Syst.*, 2020-December(NeurIPS) :1–23.

Chaumette, F. and Hutchinson, S. (2006). Visual servo control. i. basic approaches. *IEEE Robotics & Automation Magazine*, 13(4) :82–90.

Chaumette, F. and Hutchinson, S. (2007). Visual servo control. ii. advanced approaches [tutorial]. *IEEE Robotics & Automation Magazine*, 14(1) :109–118.

Chen, C., Ying, V., and Laird, D. (2016). Deep q-learning with recurrent neural networks. *stanford cs229 course report*, 4 :3.

Chen, Y., Ma, Y., and Yun, W. (2013). Application of improved genetic algorithm in pid controller parameters optimization. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 11(3) :1524–1530.

Chen, Z., Wang, J., Wang, X., Zhao, Y., Li, X., and Liu, Y. (2022). Design and control of soft biomimetic pangasius fish robot using fin ray effect and reinforcement learning. *Scientific Reports*, 12(1) :21861.

Cheng, J.-Y., Pedley, T., and Altringham, J. (1998). A continuous dynamic beam model for swimming fish. *Philosophical Transactions of the Royal Society of London. Series B : Biological Sciences*, 353(1371) :981–997.

Childress, S. (1981). *Mechanics ofswimming and flying*. Number 2. Cambridge University Press.

Clapham, R. J. and Hu, H. (2014a). ISplash-I : High performance swimming motion of a carangiform robotic fish with full-body coordination. *Proc. - IEEE Int. Conf. Robot. Autom.*, (iii) :322–327.

Clapham, R. J. and Hu, H. (2014b). ISplash-II : Realizing fast carangiform swimming to outperform a real fish. *IEEE Int. Conf. Intell. Robot. Syst.*, (Iros) :1080–1086.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elu). In *Int. Conf. Mach. Learn.*, pages 1020–1028. PMLR.

Cohen, G. and Coon, G. (1953). Theoretical consideration of retarded control. *Transactions of the American Society of Mechanical Engineers*, 75(5) :827–834.

Conniffe, D. (1988). Obtaining expected maximum log likelihood estimators. Papers me176, Economic and Social Research Institute (ESRI).

Coombs, S. and Montgomery, J. (2014). The role of flow and the lateral line in the multisensory guidance of orienting behaviors. In *Flow Sensing in Air and Water : Behavioral, Neural and Engineering Principles of Operation*, pages 65–101. Springer.

Curatolo, M. and Teresi, L. (2015). The virtual aquarium : simulations of fish swimming. In *Proc. European COMSOL Conference*.

Currier, T. M., Lheron, S., and Modarres-Sadeghi, Y. (2020). A bio-inspired robotic fish utilizes the snap-through buckling of its spine to generate accelerations of more than 20g. *Bioinspiration and Biomimetics*, 15(5).

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2017). Distributional reinforcement learning with quantile regression. *arXiv preprint arXiv :1710.10044*.

Dechter, R. (1986). Learning While Searching in Constraint-Satisfaction-Problems. In *AAAI*, pages 178–185. Morgan Kaufmann.

DeGol, J., Bretl, T., and Hoiem, D. (2017). Chromatag : A colored marker and fast detection algorithm. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1472–1481.

Deisenroth, M. P. and Rasmussen, C. E. (2011). PILCO : A model-based and data-efficient approach to policy search. *Proc. 28th Int. Conf. Mach. Learn. ICML 2011*, pages 465–472.

Devlin, K. J. (2002). The millennium problems : The seven greatest unsolved mathematical puzzles of our time.

Di Santo, V., Goerig, E., Wainwright, D. K., Akanyeti, O., Liao, J. C., Castro-Santos, T., and Lauder, G. V. (2021). Convergence of undulatory swimming kinematics across a diversity of fishes. *Proceedings of the National Academy of Sciences*, 118(49) :e2113206118.

Dietterich, T. G. (2000). *Ensemble Methods in Machine Learning*. Springer.

Dill, E. H. (2006). *Continuum mechanics : elasticity, plasticity, viscoelasticity*. CRC press.

Du, R., Li, Z., Youcef-Toumi, K., Valdivia, P., and Editors, A. (2015). *Springer Tracts in Mechanical Engineering Robot Fish Bio-inspired Fishlike Underwater Robots*.

Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12 :2121–2159.

Durand, S., Castellanos, J. F. G., Marchand, N., and Sanchez, W. F. G. (2013). Event-based control of the inverted pendulum : Swing up and stabilization. *Journal of Control Engineering and Applied Informatics*, 15(3) :96–104.

Eakins, B. and Sharman, G. (2010). Volumes of the world's oceans from etopo1 ; noaa national geophysical data center. *Boulder, 1p.[Google Scholar]*.

Epps, B., Valdivia y Alvarado, P., Youcef-Toumi, K., and H. Teche, A. (2009). Swimming performance of a biomimetic compliant fish-like robot. *Exp Fluids*, 47 :927–939.

Farhat, M., Alghamdi, A., Alotaibi, M., El-Khodary, M., and Aly, A. E. (2019). Biomimetic drones : A review of design, control, and applications. *Aerospace Science and Technology*, 94 :105–123.

Felton, S., Brault, P., Fromont, E., and Marchand, E. (2022). Visual servoing in autoencoder latent space. *IEEE Robotics and Automation Letters*, 7(2) :3234–3241.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv Prepr. arXiv1703.03400*.

Floryan, D., Van Buren, T., and Smits, A. J. (2018). Efficient Cruising for Swimming and Flying Animals Is Dictated by Fluid Drag. *Proceedings of the National Academy of Sciences*, 115(32) :8116–8118.

Fukushima, K. (1980). Neocognitron : A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biol. Cybern.*, 36(4) :193–202.

Gao, B., Guo, S., and Ye, X. (2011). Motion-control analysis of icpf-actuated underwater biomimetic microrobots. *International Journal of Mechatronics and Automation*, 1(2) :79–89.

Gayon, J. (1998). *Darwinism's Struggle for Survival : Heredity and the Hypothesis of Natural Selection*. Cambridge University Press.

Gazzola, M., Argentina, M., and Mahadevan, L. (2014). Scaling macroscopic aquatic locomotion. *Nature Physics*, 10(10) :758–761.

Gazzola, M., Argentina, M., and Mahadevan, L. (2015). Gait and speed selection in slender inertial swimmers. *Proceedings of the National Academy of Sciences*, 112(13) :3874–3879.

Geirhos, R., Janssen, D. H., Schütt, H. H., Rauber, J., Bethge, M., and Wichmann, F. A. (2017). Comparing deep neural networks against humans : object recognition when the signal gets weaker. *arXiv preprint arXiv :1706.06969*.

Ghavamzadeh, M., Engel, Y., and Valko, M. (2016). Bayesian policy gradient and actor-critic algorithms. *Journal of Machine Learning Research*, 17(66) :1–53.

Gibouin, F., Raufaste, C., Bouret, Y., and Argentina, M. (2018). Study of the thrust–drag balance with a swimming robotic fish. *Physics of Fluids*, 30(9) :091901.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. *B. Deep Learn.*

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *Adv. Neural Inf. Process. Syst.*, volume 27, pages 2672–2680.

Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9).

Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., and Levine, S. (2019a). Learning to Walk Via Deep Reinforcement Learning.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017a). Reinforcement learning with deep energy-based policies. *34th Int. Conf. Mach. Learn. ICML 2017*, 3 :2171–2186.

Haarnoja, T., Zhou, A., Gupta, A., Abbeel, P., and Levine, S. (2017b). Reinforcement Learning with Deep Energy-Based Policies. In *Proc. 34th Int. Conf. Mach. Learn.*, volume 70, pages 1905–1914.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2019b). Soft actor-critic algorithms and applications. *arXiv :1812.05905*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning latent dynamics for planning from pixels. *36th Int. Conf. Mach. Learn. ICML 2019*, 2019-June :4528–4547.

Hafner, D., Lillicrap, T. P., Sutskever, I., and Silver, D. (2020). Dreamer : Efficient meta-learning for fast adaptation to new tasks. *arXiv Prepr. arXiv2006.04868*.

Hahnloser, R. H. R., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789) :947–951.

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*.

Healy, K., Ezard, T. H., Jones, O. R., Salguero-Gómez, R., and Buckley, Y. M. (2019). Animal life history is shaped by the pace of life and the distribution of age-specific mortality and reproduction. *Nature ecology & evolution*, 3(8) :1217–1224.

Hemelrijk, C. K., Reid, D., Hildenbrandt, H., and Padding, J. (2015). The increased efficiency of fish swimming in a school. *Fish and Fisheries*, 16(3) :511–521.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *32nd AAAI Conference on Artificial Intelligence*.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2017). Rainbow : Combining improvements in deep reinforcement learning. *arXiv preprint arXiv :1710.02298*.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). B-VAE : Learning basic visual concepts with a constrained variational framework. *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, pages 1–13.

Hill, A. V. (1938). The heat of shortening and the dynamic constants of muscle. *Proceedings of the Royal Society of London. Series B-Biological Sciences*, 126(843) :136–195.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1990). Connectionist Architectures for Natural Language Processing. In *Parallel Distrib. Process. Explor. Microstruct. Cogn.*, volume 2, pages 77–109. MIT Press.

Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. (1984). Boltzmann Machines : Constraint Satisfaction Networks that Learn. Technical Report CMU-CS-84-119, Carnegie Mellon University.

Hiraoka, T., Imagawa, T., Hashimoto, T., Onishi, T., and Tsuruoka, Y. (2021). Dropout q-functions for doubly efficient reinforcement learning. *arXiv preprint arXiv :2110.02034*.

Hirt, M. R., Jetz, W., Rall, B. C., and Brose, U. (2017). A general scaling law reveals why the largest animals are not the fastest. *Nature Ecology & Evolution*, 1(8) :1116–1122.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8) :1735–1780.

Hoover, A. P., Cortez, R., Tytell, E. D., and Fauci, L. J. (2018). Swimming performance, resonance and shape evolution in heaving flexible panels. *Journal of Fluid Mechanics*, 847 :386–416.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5) :359–366.

Hu, H., Oyekan, J., and Gu, D. (2011). *A School of Robotic Fish for Pollution Detection in Port*, pages 85–104.

Hu, Y., Zhao, W., Wang, L., and Jia, Y. (2009a). Underwater target following with a vision-based autonomous robotic fish.

Hu, Y., Zhao, W., Xie, G., and Wang, L. (2009b). Development and target following of vision-based autonomous robotic fish. *Robotica*, 27(7) :1075–1089.

Huang, J., Ding, F., Fukuda, T., and Matsuno, T. (2012). Modeling and velocity control for a novel narrow vehicle based on mobile wheeled inverted pendulum. *IEEE Transactions on Control Systems Technology*, 21(5) :1607–1617.

Hunter, J. (1971). Swimming speed, tail beat frequency, tail beat amplitude and size in jack mackerel, trachurus symmetricus, and other fishes. *Fish. Bull.*, 69 :253–266.

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots : a review. *Neural Networks*, 21(4) :642–653.

Israilov, S., Fu, L., Sánchez-Rodríguez, J., Fusco, F., Allibert, G., Raufaste, C., and Argentina, M. (2023). Pendule Pi : A repository for controlling the pendulum using DRL approaches. `https://github.com/ss555/cart_pole`. All the codes described in the manuscript are open-source and available at `https://github.com/ss555/cart_pole`. A reference manual is also published at `francofusco.github.io/pendule_pi` to assist teachers and students during the first-time hardware-setup.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to Trust Your Model : Model-Based Policy Optimization. *arXiv Prepr. arXiv1906.08253*.

Ji, D., ur Rehman, F., Ajwad, S. A., Shahani, K., Sharma, S., Sutton, R., Li, S., Ye, Z., Zhu, H., and Zhu, S. (2020). Design and development of autonomous robotic fish for object detection and tracking. *Int. J. Adv. Robot. Syst.*, 17(3) :1–11.

Jiao, Y., Ling, F., Heydari, S., Heess, N., Merel, J., and Kanso, E. (2021). Learning to swim in potential flow. *Physical Review Fluids*, 6(5) :050505.

Jumper, J. J., Evans, R., Pritzel, A., Green, T., Figurnov, G., Wang, T., Wu, T., Ho, S. Y., Kohli, P., Chen, Y., and Others (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873) :582–588.

Kailath, T. (1980). *Linear systems*. Prentice-Hall.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

Katzschmann, R. K., DelPreto, J., MacCurdy, R., and Rus, D. (2018). Exploration of underwater life with an acoustically controlled soft robotic fish. *Sci. Robot.*, 3(16) :1–13.

Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., and Scaramuzza, D. (2023). Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976) :982–987.

Kelasidi, E., Elgenes, G., and Kilvær, H. (2018). Fluid parameter identification for underwater snake robots. In *ASME 2018 37th International Conference on Ocean, Offshore and Arctic Engineering*, pages V07AT06A018–V07AT06A018. American Society of Mechanical Engineers.

Kelasidi, E., Liljeback, P., Pettersen, K. Y., and Gravdahl, J. T. (2016). Innovation in underwater robots : biologically inspired swimming snake robots. *IEEE robotics & automation magazine*, 23(1) :44–62.

Khalil, W. and Dombre, E. (2002). *Modeling, Identification and Control of Robots*. Butterworth-Heinemann, Oxford.

Kim, H., Jordan, M., Sastry, S., and Ng, A. (2004). Autonomous helicopter flight via reinforcement learning. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press.

Kingma, D. P. and Ba, J. (2014). Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*.

Kingma, D. P. and Welling, M. (2014a). Auto-encoding variational bayes. *2nd Int. Conf. Learn. Represent. ICLR 2014 - Conf. Track Proc.*, (Ml) :1–14.

Kingma, D. P. and Welling, M. (2014b). Stochastic gradient vb and the variational auto-encoder. In *Second international conference on learning representations, ICLR*, volume 19, page 121.

Kirk, D. E. (2004). *Optimal control theory : an introduction*. Courier Corporation.

Kohnen, W. (2009). Human exploration of the deep seas : Fifty years and the inspiration continues. *Marine Technology Society Journal*, 43(5) :42–62.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Adv. Neural Inf. Process. Syst.*, volume 12, pages 1008–1014. MIT Press.

Koo, B., Kim, D. Y., Park, J.-H., Kim, J., Kim, K. M., and Park, W.-J. (2018). Biomimetic bipedal robots : A review of recent advances and future challenges. *IEEE Robotics and Automation Magazine*, 25(2) :92–105.

Koryakovskiy, I., Kudruss, M., Babuška, R., Caarls, W., Kirches, C., Mombaur, K., Schlöder, J. P., and Vallery, H. (2017). Benchmarking model-free and model-based optimal control. *Robotics and Autonomous Systems*, 92 :81–90.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *Neural Computation*, 3(1) :132–144.

Krause, J., Winfield, A., and Deneubourg, J. (2011). Interactive robots in experimental biology. *Trends in Ecology and Evolution*, 26(6) :369–375.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.*, 25 :1097–1105.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6) :84–90.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33 :1179–1191.

Kumar, S. (2020). Balancing a cartpole system with reinforcement learning–a tutorial. *arXiv preprint arXiv :2006.04938*.

Kuznetsov, Y. A., Kuznetsov, I. A., and Kuznetsov, Y. (1998). *Elements of applied bifurcation theory*, volume 112. Springer.

Landau, L. and Lifshitz, E. (1987). *Fluid mechanics*, volume 6 of *Course of Theoretical Physics*.

Landau, L. D., Lifšic, E. M., Lifshitz, E. M., Kosevich, A. M., and Pitaevskii, L. P. (1986). *Theory of elasticity : volume 7*, volume 7. Elsevier.

Lauder, G. V. and Tytell, E. D. (2005). Hydrodynamics of undulatory propulsion. *Fish physiology*, 23 :425–468.

Lazarini, A. Z. N., de Souza Ribeiro, J. M., and Jorgetto, M. F. C. (2014). Low cost implementation of a inverted pendulum control system. In *11th IEEE/IAS International Conference on Industry Applications*, pages 1–5.

Lebastard, V., Boyer, F., and Lanneau, S. (2016). Reactive underwater object inspection based on artificial electric sense. *Bioinspir. Biomim.*, 11(4) :045003.

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10) :1995.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553) :436–444.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11) :2278–2324.

LeCun, Y., Bottou, L., Orr, G., and Muller, X. (1998). Efficient backprop. *Neural Comput.*, 10(2) :215–244.

Lee, A. X., Levine, S., and Abbeel, P. (2017). Learning visual servoing with deep features and fitted q-iteration. *arXiv preprint arXiv :1703.11000*.

Lee, G. H. and Jung, S. (2008). Design and control of an inverted pendulum system for intelligent mechatronics system control education. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1254–1259.

Lee, K. Y., Park, S.-J., Matthews, D. G., Kim, S. L., Marquez, C. A., Zimmerman, J. F., Ardoña, H. A. M., Kleber, A. G., Lauder, G. V., and Parker, K. K. (2022). An autonomously swimming biohybrid fish designed with human cardiac biophysics. *Science*, 375(6581) :639–647.

Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., and Filliat, D. (2018). State representation learning for control : An overview. *Neural Networks*, 108 :379–392.

Li, G., Ashraf, I., François, B., Kolomenskiy, D., Lechenault, F., Godoy-Diana, R., and Thiria, B. (2021a). Burst-and-Coast Swimmers Optimize Gait by Adapting Unique Intrinsic Cycle. *Communications Biology*, 4(1) :1–7.

Li, H. and Zhou, Z. (2013). State aggregation for optimal control of large discrete-time markov chains. *Automatica*, 49(7) :1868–1877.

Li, Y., Liu, Y., Zhang, J., and Yu, J. (2021b). Using a robotic platform to study the influence of relative tailbeat phase on the energetic costs of side-by-side swimming in fish. *Proc. R. Soc. A*, 477(2240) :20200810.

Liao, J. C. (2007). A review of fish swimming mechanics and behaviour in altered flows. *Philosophical Transactions of the Royal Society B : Biological Sciences*, 362(1487) :1973–1993.

Lighthill, M. J. (1971). Large-amplitude elongated-body theory of fish locomotion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 179 :125–138.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*

Liu, M., Zhu, M., and Zhang, W. (2022a). Goal-conditioned reinforcement learning : Problems and solutions. *arXiv preprint arXiv :2201.08299*.

Liu, W., Bai, K., He, X., Song, S., Zheng, C., and Liu, X. (2022b). Fishgym : A high-performance physics-based simulation framework for underwater robot learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6268–6275.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. (2022c). A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986.

Loaiza-Ganem, G. and Cunningham, J. P. (2019). The continuous bernoulli : fixing a pervasive error in variational autoencoders. *Advances in Neural Information Processing Systems*, 32.

Lubchenco, J. and Haugan, P. M. (2023). Technology, data and new models for sustainably managing ocean resources. In *The Blue Compendium : From Knowledge to Action for a Sustainable Ocean Economy*, pages 185–211. Springer International Publishing.

Lundberg, K. H. and Barton, T. W. (2010). History of Inverted-Pendulum Systems. *IFAC Proceedings Volumes*, 42(24) :131–135.

Lyapunov, A. M. (1892). The general problem of motion stability. *Annals of Mathematics Studies*, 17(1892).

Madgwick, S. O., Harrison, A. J., and Vaidyanathan, R. (2011). Estimation of IMU and MARG orientation using a gradient descent algorithm. In *2011 IEEE international conference on rehabilitation robotics*, pages 1–7.

Mahony, R., Corke, P., and Hamel, T. (2007). Dynamic Image-Based Visual Servo Control Using Centroid and Optic Flow Features. *Journal of Dynamic Systems, Measurement, and Control*, 130(1) :011005.

Manrique Escobar, C. A., Pappalardo, C. M., and Guida, D. (2020). A parametric study of a deep reinforcement learning control system applied to the swing-up problem of the cart-pole. *Applied Sciences*, 10(24).

Marras, S., Buitrago, A. C., and Porfiri, M. (2012). Fish and robots swimming together : attraction towards the robot demands biomimetic locomotion. *Journal of The Royal Society Interface*, 8(64) :849–860.

Martins, R. V. A., Martins, J. M. S., and Costa, M. M. (2017). Biomimetic swimming propulsion : An overview of design principles and engineering challenges. *Smart Mater. Struct.*, 26(12) :125031.

Mathis, A., Schneider, S., Lauer, J., and Mathis, M. W. (2020). A primer on motion capture with deep learning : Principles, pitfalls, and perspectives. *Neuron*, 108 :44–65.

Mawhin, J. (2005). *Alexandr Mikhailovich Lyapunov, thesis on the stability of motion (1892)*, pages 664–676.

Michelin, S. and Llewellyn Smith, S. G. (2009). Resonance and propulsion performance of a heaving flexible wing. *Physics of Fluids*, 21(7) :071902.

Mitin, I. and Lobov, S. A. (2022). Bioinspired propulsion system for a thunniform robotic fish. *Biomimetics*, 7(4) :215.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv :1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., 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(7540) :529–533.

Muhammad, U., Wang, W., Chattha, S. P., and Ali, S. (2018). Pre-trained VGGNet architecture for remote-sensing image scene classification. In *IEEE 24th International Conference on Pattern Recognition (ICPR)*, pages 1622–1627.

Nair, A., Gupta, A., Dalal, M., and Levine, S. (2020). Awac : Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv :2006.09359*.

Nair, A., Pong, V., Dalal, M., Bahl, S., and Levine, S. (2018). Visual reinforcement learning with imagined goals. In *Neural Information Processing Systems*, pages 9110–9120.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2006). Autonomous inverted helicopter flight via reinforcement earning. *Springer Tracts Adv. Robot.*

Novati, G., Mahadevan, L., and Koumoutsakos, P. (2019). Controlled gliding and perching through deep-reinforcement-learning. *Physical Review Fluids*, 4(9).

Oppenheim, A. V., Schafer, R. W., and Buck, J. R. (1999). *Signals and systems*. Pearson Education.

Özalp, R., Varol, N. K., Taşci, B., and Uçar, A. (2020). *A Review of Deep Reinforcement Learning Algorithms and Comparative Results on Inverted Pendulum System*, pages 237–256. Springer International Publishing.

Paraz, F., Schouveiler, L., and Eloy, C. (2016). Thrust generation by a heaving flexible foil : Resonance, nonlinearities, and optimality. *Physics of Fluids*, 28(1) :011903.

Pearson, K. G. (1995). Proprioceptive regulation of locomotion. *Current opinion in neurobiology*, 5(6) :786–791.

Pedley, T. and Hill, S. (1999). Large-amplitude undulatory fish swimming : fluid mechanics coupled to internal mechanics. *Journal of Experimental Biology*, 202(23) :3431–3438.

Pedley, T. J. (1977). Scale effects in animal locomotion. In *International Symposium on Scale Effects in Animal Locomotion (1975 : Cambridge University)*. Academic Press.

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4) :682–697.

Polverino, G., Phamduy, P., and Porfiri, M. (2013). Fish and Robots Swimming Together in a Water Tunnel : Robot Color and Tail-Beat Frequency Influence Fish Behavior. *PLoS One*, 8(10) :47–50.

Pomerleau, D. (1988). Alvinn : An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313.

Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. (2019). Skew-fit : State-covering self-supervised reinforcement learning. *arXiv preprint arXiv :1903.03698*.

Pontryagin, L. S. (1987). *Mathematical theory of optimal processes*. CRC press.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. (2019a). Stable baselines3. `https://stable-baselines3.readthedocs.io/en/master/`.

Raffin, A., Hill, A., Traoré, R., Lesort, T., Díaz-Rodríguez, N., and Filliat, D. (2019b). Decoupling feature extraction from policy learning : assessing benefits of state representation learning in goal based robotics. *arXiv preprint arXiv :1901.08651*.

Rajendran, S. K. and Zhang, F. (2022). Design, Modeling, and Visual Learning-Based Control of Soft Robotic Fish Driven by Super-Coiled Polymers. *Frontiers in Robotics and AI*, 8.

Ramananarivo, S. (2014). *Propulsion biomimétique de structures élas-tiques*. PhD thesis, Université Paris-Diderot-Paris VII.

Ramananarivo, S., Godoy-Diana, R., and Thiria, B. (2013). Passive elastic mechanism to mimic fish-muscle action in anguilliform swimming. *Journal of The Royal Society Interface*, 10(88) :20130667.

Ramponi, M. (2022). How chatgpt actually works.

Riedmiller, M. (2005). Neural Reinforcement Learning to Swing-Up and Balance a Real Pole. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 3191–3196 Vol. 4.

Rohr, J. J. and Fish, F. E. (2004). Strouhal numbers and optimization of swimming by odontocete cetaceans. *Journal of Experimental Biology*, 207(10) :1633–1642.

Rosenblatt, F. (1958). The Perceptron : A probabilistic model for information storage and retrieval. *Psychol. Rev.*, 65(6) :386–408.

Ross, S., Gordon, A., and Silver, D. (2011). DAgger : Efficient Learning from Demonstrations. *Conf. Robot Learn.*

Ruckstiess, T., Felder, M., and Schmidhuber, J. (2008). State-dependent exploration for policy gradient methods. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 5212 LNAI(PART 2) :234–249.

Saadat, M., Fish, F. E., Domel, A., Di Santo, V., Lauder, G., and Haj-Hariri, H. (2017). On the rules for aquatic locomotion. *Physical Review Fluids*, 2(8) :083102.

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM J. Res. Dev.*, 3(3) :210–229.

Sanchez, C., Arribart, H., and Guille, M. M. G. (2005). Biomimetism and bioinspiration as tools for the design of innovative materials and systems. *Nature Materials*, 4(4) :277–288.

Sánchez-Rodríguez, J. (2021). *Mécanismes de locomotion ondulatoire sous-marine*. PhD thesis, Université Côte d'Azur.

Sánchez-Rodríguez, J., Celestini, F., Raufaste, C., and Argentina, M. (2021). Proprioceptive mechanism for bioinspired fish swimming. *Phys. Rev. Lett.*, 126 :234501.

Sánchez-Rodríguez, J., Raufaste, C., and Argentina, M. (2020). A minimal model of self propelled locomotion. *Journal of Fluids and Structures*, 97 :103071.

Sánchez-Rodríguez, J., Raufaste, C., and Argentina, M. (2023). Scaling the tail beat frequency and swimming speed in underwater undulatory swimming. *Nature Communications*, 14(1) :5569.

Sandha, S. S., Garcia, L., Balaji, B., Anwar, F., and Srivastava, M. (2021). Sim2real transfer for deep reinforcement learning with stochastic state transition delays. pages 1066–1083.

Santo, V. D., Goerig, E., Wainwright, D. K., Akanyeti, O., Liao, J. C., Castro-Santos, T., and Lauder, G. V. (2021). Convergence of undulatory swimming kinematics across a diversity of fishes. *Proceedings of the National Academy of Sciences*, 118(49) :e2113206118.

Schulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P. (2015). Trust region policy optimization. *32nd Int. Conf. Mach. Learn. ICML 2015*, 3 :1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, pages 1–14.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv :1707.06347*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G. d., Schrittwieser, J., Antonoglou, I., Panneerselvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587) :484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, 550 :354–359.

Sonneborn, L. and Van Vleck, F. (1964). The bang-bang principle for linear control systems. *Journal of the Society for Industrial and Applied Mathematics, Series A : Control*, 2(2) :151–159.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout : a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1) :1929–1958.

Sugihara, T., Nakamura, Y., and Inoue, H. (2002). Real-time humanoid motion generation through ZMP manipulation based on inverted pendulum control. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1404–1409.

Sun, W., Su, S.-F., Xia, J., and Wu, Y. (2018). Adaptive tracking control of wheeled inverted pendulums with periodic disturbances. *IEEE Transactions on Cybernetics*, 50(5) :1867–1876.

Surriani, A., Wahyunggoro, O., and Cahyadi, A. I. (2021). Reinforcement learning for cart pole inverted pendulum system. In *2021 IEEE Industrial Electronics and Applications Conference*, pages 297–301.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning : An introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T. P., and Riedmiller, M. A. (2018). Deepmind control suite. *CoRR*, abs/1801.00690.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3) :58–68.

Thandiackal, R., Melo, K., Paez, L., Herault, J., Kano, T., Akiyama, K., Boyer, F., Ryczko, D., Ishiguro, A., and Ijspeert, A. J. (2021). Emergence of robust self-organized undulatory swimming based on local hydrodynamic force sensing. *Science Robotics*, 6(57).

Todorov, E. (2014). Convex and analytically-invertible dynamics with contacts and constraints : Theory and implementation in mujoco. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6054–6061.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco : A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023). Llama : Open and efficient foundation language models. *arXiv preprint arXiv :2302.13971*.

Triantafyllou, G. S., Triantafyllou, M. S., and Grosenbaugh, M. A. (1993). Optimal thrust development in oscillating foils with application to fish propulsion. *Journal of Fluids and Structures*, 7(2) :205–224.

Triantafyllou, M. S. and Triantafyllou, G. S. (1995). An efficient swimming machine. *Scientific American*, 272(3) :64–70.

Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., Heess, N., and Tassa, Y. (2020). dm control : Software and tasks for continuous control.

Van Buren, T., Floryan, D., and Smits, A. J. (2019). Bio-inspired underwater propulsors. In Daniel, L. and Soboyejo, W., editors, *Bioinspired Design*, chapter 11, pages 237–266. Cambridge University Press.

Van Hasselt, H., Guez, A., and Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning. *arXiv Prepr. arXiv1509.06461*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is All You Need. In *Adv. Neural Inf. Process. Syst.*, pages 5998–6008.

Vatanabe, S., Pires, R., Nakasone, P., and Silva, E. (2008). New configurations of oscillatory flow pumps using bimorph piezoelectric actuators. *Proc SPIE*, 6930.

Videler, J. J. (1993). *Fish Swimming*. Springer Netherlands.

Vogel, S. (2020). *Life in Moving Fluids : The Physical Biology of Flow-Revised and Expanded Second Edition*. Princeton University Press.

Wang, M., Dong, H., Li, X., Zhang, Y., and Yu, J. (2019). Control and Optimization of a Bionic Robotic Fish Through a Combination of CPG model and PSO. *Neurocomputing*, 337 :144–152.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.d. thesis, Cambridge University.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8 :279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8 :229–256.

Williams IV, R., Neubarth, N., and Hale, M. E. (2013). The function of fin rays as proprioceptive sensors in fish. *Nature communications*, 4(1) :1729.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. (2021). Mastering visual continuous control : Improved data-augmented reinforcement learning. *arXiv preprint arXiv :2107.09645*.

Youssef, S. M., Soliman, M., Saleh, M. A., Elsayed, A. H., and Radwan, A. G. (2022). Design and control of soft biomimetic pangasius fish robot using fin ray effect and reinforcement learning. *Scientific Reports*, 12(1).

Yu, J., Tan, M., Chen, J., and Zhang, J. (2014a). A survey on CPG-inspired control models and system implementation. *IEEE Trans. Neural Networks Learn. Syst.*, 25(3) :441–456.

Yu, J., Tan, M., Wang, S., and Chen, E. (2004). Development of a Biomimetic Robotic Fish and Its Control Algorithm. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 34(4) :1798–1810.

Yu, J., Wang, K., Tan, M., and Zhang, J. (2014b). Design and control of an embedded vision guided robotic fish with multiple control surfaces. *ScientificWorldJournal*, 2014 :631296.

Yu, J., Zhang, C., and Liu, L. (2016). Design and control of a single-motor-actuated robotic fish capable of fast swimming and maneuverability. *IEEE/ASME Trans. Mechatronics*, 21(3) :1711–1719.

Zhang, F., Ennasr, O., Litchman, E., and Tan, X. (2016). Autonomous Sampling of Water Columns Using Gliding Robotic Fish : Algorithms and Harmful-Algae-Sampling Experiments. *IEEE Systems Journal*, 10(3) :1271–1281.

Zhang, F., Lagor, F. D., Yeo, D., Washington, P., and Paley, D. A. (2015). Distributed flow sensing for closed-loop speed control of a flexible fish robot. *Bioinspiration & Biomimetics*, 10(6).

Zhang, S., Yan, W., and Xie, G. (2017). Consensus-based leader-following formation control for a group of semi-biomimetic robotic fishes. *Int. J. Adv. Robot. Syst.*, 14(4) :1–8.

Zhang, T., Tian, R., Wang, C., and Xie, G. (2020). Path-following control of fish-like robots : A deep reinforcement learning approach. *IFAC-PapersOnLine*, 53(2) :8163–8168.

Zhang, Z., Yang, T., Zhang, T., Zhou, F., Cen, N., Li, T., and Xie, G. (2021). Global Vision-Based Formation Control of Soft Robotic Fish Swarm. *Soft Robot.*, 8(3) :310–318.

Zheng, J., Zhang, T., Wang, C., Xiong, M., and Xie, G. (2021). Learning for Attitude Holding of a Robotic Fish : An End-to-End Approach With Sim-to-Real Transfer. *IEEE Trans. Robot.*, PP :1–17.

Zheng, Y., Li, X., and Xu, L. (2020). Balance control for the first-order inverted pendulum based on the advantage actor-critic algorithm. *International Journal of Control, Automation and Systems*, 18(12) :3093–3100.

Zhong, Y., Li, Z., and Du, R. (2017). A Novel Robot Fish with Wire-Driven Active Body and Compliant Tail. *IEEE/ASME Trans. Mechatronics*, 22(4) :1633–1643.

Zhu, J., White, C., Wainwright, D. K., Di Santo, V., Lauder, G. V., and Bart-Smith, H. (2019). Tuna robotics : A high-frequency experimental platform exploring the performance space of swimming fishes. *Science Robotics*, 4(34).

Ziegler, J. G. and Nichols, N. B. (1942). Optimum settings for automatic controllers. *Transactions of the American society of mechanical engineers*, 64(8) :759–765.

Annexes

ANNEXES A

Inverted Pendulum

A.1 Low-level Interface (LLI)

At each major control cycle, the LLI processes the raw measurements from the encoders by smoothing them with a digital 4th-order Butterworth filter [Khalil and Dombre, 2002] and by differentiating them numerically in order to estimate $\dot{x}$ and $\dot{\theta}$. For the communication, we use the ZeroMQ [1] library. This allows to write client controller applications that do not need to focus on the low-level management of hardware resources. In addition, clients can be run either from the Raspberry Pi 4 or from any other machine that is able to connect to the board, for example via the local network or via WiFi. This opens the possibility to write client applications in potentially any programming language supported by ZeroMQ. Our client applications are written in Python and C++.

A.2 Measurements of the physical parameters

The values of the physical parameters of the cart-pole are displayed in Table A.1. The pendulum mass was measured with a scale. The natural frequency ω and viscous friction coefficient k_v were inferred from the signal $\theta(t)$ of the free oscillations of the pendulum with a blocked cart as expected by eq. Eq. (2.21) with $\ddot{x} = 0$. We show in fig. A.1a, the relaxation dynamics of the pendulum, as well as the result of the numerical prediction of the model with the best fitted parameters. The parameters τ, f_c, f_d and k_U in Eq. (2.22) and Eq. (2.23) are inferred by imposing step functions as voltages and measuring the cart velocity as a function of time. Again, parameters are deduced by the best interpolations (Fig. A.1b). In Fig. A.1c, we observe in more details the effect of the three parameters f_c, f_d and k_U on the discontinuity on the velocity-axis, the up-down asymmetry and the slope respectively, while plotting the steady state velocity as a function of the applied voltage. The uncertainty on the angular velocity $\dot{\theta}$ is correlated to σ_θ and to the time resolution $\Delta t \simeq 0.05\,\mathrm{s}$. This gives an uncertainty $\sigma_{\dot{\theta}} = \sigma_\theta / \Delta t \simeq 52\,\mathrm{mrad\,s^{-1}}$.

A.3 Methodology for training RL agents

All the simulations and experiments were driven by a Dell Precision 7550 using its internal GPU. For one simulation with $15\ 10^5$ time steps with logging and evaluation loops, it takes 10.7 minutes using GPU (NVidia Quadro T2000), and 13.43 minutes using CPU only (Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz).

1. ZeroMQ : `https://zeromq.org/` (accessed on July 23rd, 2021).

Name	Value
Mass of the pendulum (m)	$0.075\,\mathrm{kg}$
Natural frequency of the pendulum (ω)	$4.882\,\mathrm{rad\,s^{-1}}$
Viscous friction coefficient of the pendulum (k_v)	$0.07\,\mathrm{N\,s\,rad^{-1}}$
Electro-mechanical time constant(τ)	$0.0482\,\mathrm{s}$
Static gain of the motor (k_U)	$0.051\,\mathrm{m\,s^{-1}\,V^{-1}}$
Static friction coefficient of the cart per unit mass (f_c)	$1.166\,\mathrm{N\,kg^{-1}}$
Static offset per unit mass (f_d)	$-0.097\,\mathrm{m\,s^{-2}}$

Table A.1 – Measured physical parameters.

A.4 Hyperparameters for RL training in experiment

	Q-learning	DQN
Learning rate (α)	0.01	0.0003
Exploration ratio (ϵ)	ϵ varies following Eq. A.1 and $\epsilon_{min} = 0.1$	0.178
Discount factor (γ)	0.99	0.995
Decay factor (d)	$10^3 - 10^6$	N/A
Buffer size	N/A	50000
Batch size	N/A	1024
Network architecture	N/A	2 hidden layers with 256 neurons
ANN optimizer	N/A	Adam [Kingma and Ba, 2014] with default parameters
Loss type	N/A	Huber
Activation function	N/A	Rectified Linear Unit (ReLU)
Target update interval (C)	N/A	1000
Train frequency	1 step	1 episode
Gradient steps	N/A	as many as there were steps since last neural net update

Table A.2 – Hyperparameters for Q-learning and DQN.

Q-learning The hyperparameters for Q-learning were set as follows. We set $\alpha = 0.01$ in Eq. (2.43). As for the hyperparameter ϵ (ϵ-greedy policy), it is a good practice to promote the exploration in the early stage of the learning process with ϵ close to 1, while a small ϵ helps to

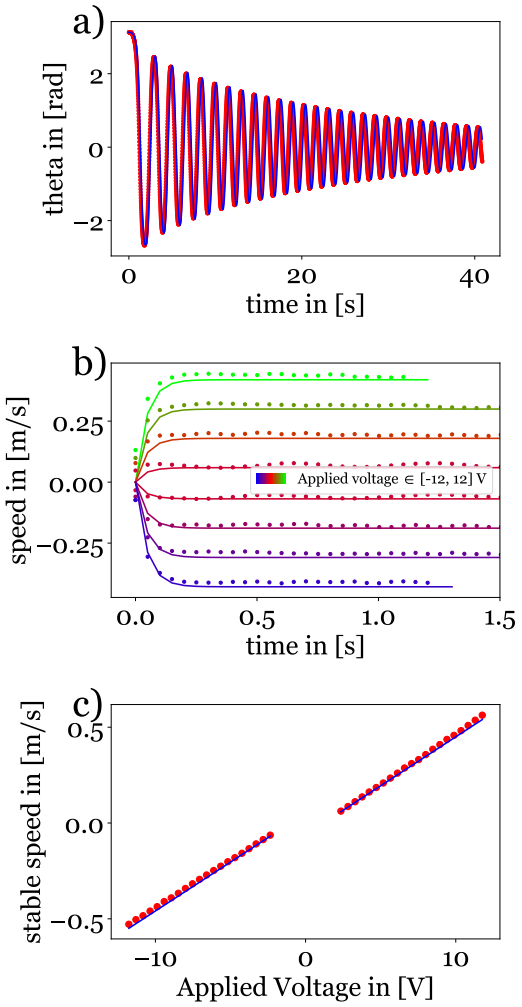

Figure A.1 – Determination of the physical parameters from interpolations (solid blue curves) of experimental data (red dots). a) Angle evolution of a free oscillation of the pendulum. b) Cart's velocity evolution with different voltages. After a short transition period, the cart's velocity reaches a plateau for all the voltages. c) Linear dependence of the plateau values on the applied voltages.

converge quickly at the end of the process. Here ϵ decreases as a function of time :

$$\epsilon = \max(\epsilon_{min}, \min(1, 1 - \log_{10}((n+1)/d)), \tag{A.1}$$

where n is the number of current episode. The decay coefficient d and ϵ_{min} are hyperparameters that can be tuned ; in this work we took $d = N_T/10$ and $\epsilon_{min} = 0.1$.

DQN The parameters were tuned with the help of Optuna [Akiba et al., 2019] framework. We discovered that the most sensible hyperparameters are network architecture, batch size and exploration rate. The complete tuning focused on the following parameters :

— **Buffer size** : the size of a buffer with transitions $(\cos(\theta), \sin(\theta), \dot{\theta}, x, \dot{x})$ used for learning the weights of the policy.
— **Batch size** : number of samples used for the gradient descent update of neural network. In practice it should be large enough to avoid biased experience, but not too large to slow the learning.
— **Learning rate** : an extent at which we update the “state-value" function at each step.
— **Gamma** : discount rate of future steps, which tells how the present is more valuable than the future.
— **Exploration rate** : the rate at which an agent explores (acts randomly) in the environment
— **Network size** : size of the dense neural network for $Q(s, a)$ function approximation
— **Target update interval** : it is the interval after which we update the target. In general, the bigger the value, the more stable is the training, but decreases the learning speed
— **Train frequency** : the frequency of learning the weights from the experience ; in our case we train the neural network at the end of every episode, since it is the most suitable way to be applied on real-life robotic reinforcement learning.

Hyperparameters for different RL simulations of the cart-pole system

Parameter	dqn&ddqn	ppo	sac	ppo continous
Discount factor (γ)	0.995	0.95	0.99	0.98
Learning rate (α)	0.0003	0.0017442	0.001	0.00025554
Exploration (ϵ)	0.17788	N/A	N/A	N/A
Target update interval (C)	1000	N/A	N/A	N/A
Buffer size (N)	50000	2048	300000	2048
Train frequency	every episode	N/A	every episode	N/A
Gradient steps	-1	N/A	-1	N/A
Batch size	1024	256	1024	8
Entropy coefficient	N/A	2.6481e-08	auto	0.077322
Policy update clip range	N/A	0.4	N/A	0.1
N epochs	N/A	20	N/A	20
λ in GAE(γ, λ)	N/A	0.9	N/A	0.8
Max Grad Norm	N/A	1	N/A	0.7
c_v	N/A	0.35163	N/A	0.69625
Moving average τ	N/A	N/A	0.02	N/A

Table A.3 – Hyperparameters list for different deep RL algorithms applied on cart-pole problem. The hyperparameters are not detailed for simplicity [more details for PPO in [Schulman et al., 2017]].

A.5 Parameters for model based control of real cart-pole

Name	Value
kp	-127.458
kpd	-822.638
kt	2234.65
ktd	437.117

Table A.4 – LQR parameters used to stabilize an inverted pendulum in unstable equilibrium in experiments.

A.6 Raw training curves : DQN and Q-learning

Raw learning curves of DQN in simulations and experiments. Sudden negative peaks in Fig. A.2 during learning correspond to expolration episodes, where the agent acts suboptimally to find new strategies to maximize the reward.

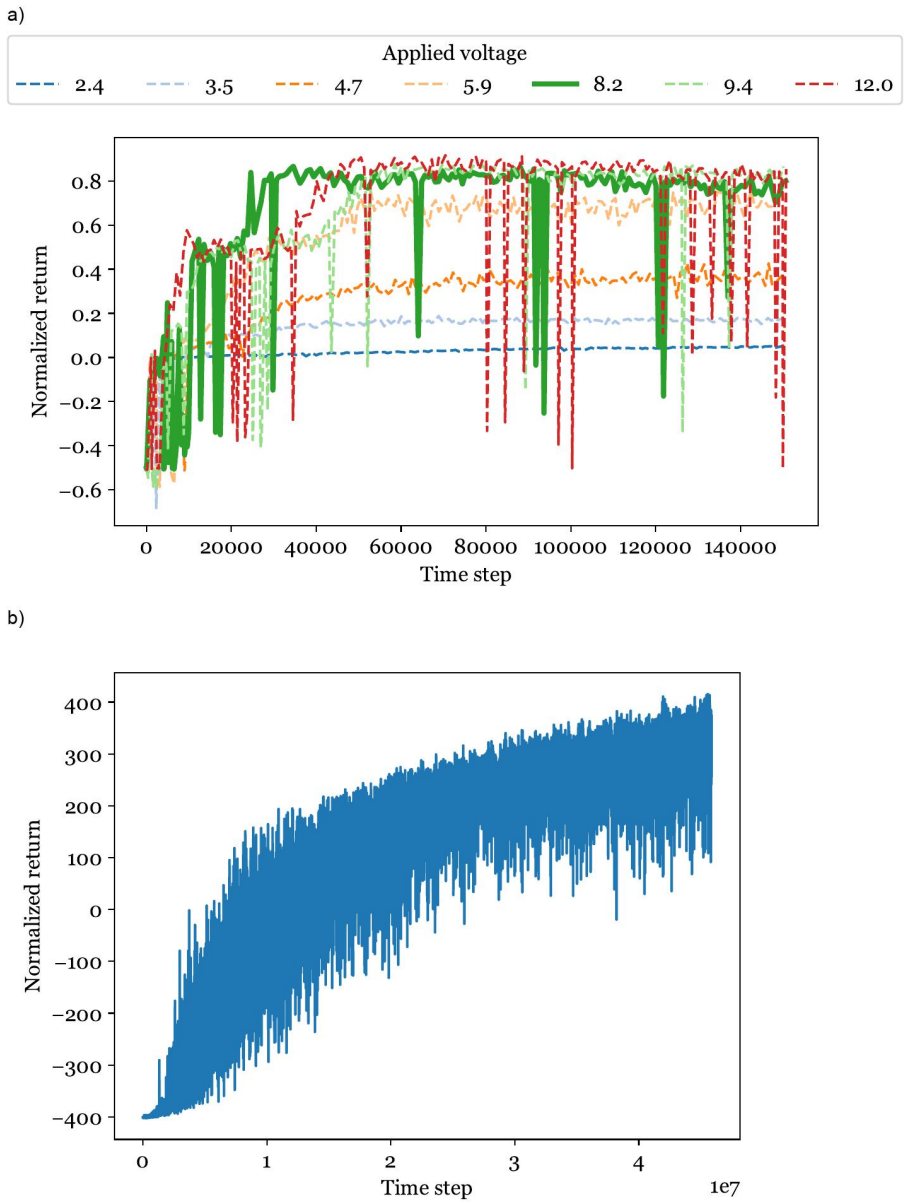

Figure A.2 – Raw learning curves without moving average : a) DQN for different voltages b) Q-learning in simulation with averaging on 10 episodes. Without averaging the figure becomes too noisy.

A.7 Cart-pole stabilization using bang-bang control

We give the simple example of bang-bang controller application on a cart-pole system described before in Eq. (2.21). The objective is to stabilize the pendulum at its unstable equilibrium ($\theta = \pi$), governed by the following ODE :

$$\ddot{\theta} + k_v\dot{\theta} + \omega_0^2 \sin\theta + \frac{\ddot{x}}{l}\cos\theta = 0$$

After the linearization at ($\theta = \pi + \beta$, where $\beta \ll 1$), we obtain :

$$\ddot{\beta} + k_v\dot{\beta} - \omega_0^2\beta - \frac{\ddot{x}}{l} = 0$$

We want to minimize the following objective :

$$I = \frac{1}{T}\int_0^T \beta^2 dt \tag{A.2}$$

under the dynamics :

$$\begin{aligned} \dot{\beta} &= \omega \\ \dot{\omega} &= -k_v\omega + \omega_0^2\beta + \frac{\ddot{x}}{l} \end{aligned} \tag{A.3}$$

Within the variational methods framework, we write the Lagrangian :

$$L = \beta^2 + \lambda_1(\dot{\beta} - \omega) + \lambda_2\left(\dot{\omega} + k_v\omega - \omega_0^2\beta - \frac{\ddot{x}}{l}\right) \tag{A.4}$$

The Hamiltonian can be written in the following form :

$$\begin{aligned} H &= -L + \lambda_1\dot{\beta} + \lambda_2\dot{w} \\ &= -\beta^2 + \lambda_1 w + \lambda_2\left(-k_v w + \omega_0^2\beta + \frac{\ddot{x}}{l}\right) \end{aligned} \tag{A.5}$$

The costate equations are defined as :

$$\begin{aligned} \dot{\lambda}_1 &= -\partial_\beta H = \lambda_2\omega_0^2 \\ \dot{\lambda}_2 &= -\partial_\omega H = -k_v\lambda_2 - \lambda_1. \end{aligned} \tag{A.6}$$

By differentiating Eq. (A.6), we obtain :

$$\ddot{\lambda}_2 = -k_v\dot{\lambda}_2 - \lambda_2\omega_0^2 \tag{A.7}$$

As $\ddot{x} \in [-\Gamma, \Gamma]$, then in order to maximize the Hamiltonian, from Eq. (A.5) we conclude that $\ddot{x} = sign(\lambda_2)\Gamma$. We thus obtain the optimal policy :

$$\ddot{\beta} + k_v\dot{\beta} - \omega_0^2\beta - \frac{\Gamma}{l} = 0. \quad 0 < t < \frac{T}{2}, \tag{A.8}$$

subject to boundary conditions (we explicitly choose these periodic boundary conditions that we will observe with the swimming fish) :

$$\begin{aligned} \beta(0) &= \beta\left(\frac{T}{2}\right) \\ \dot{\beta}(0) &= -\dot{\beta}\left(\frac{T}{2}\right) \end{aligned}$$

The solution to the above ODE is :

$$\beta(t) = -\frac{\Gamma}{l\omega_0^2} + \frac{\Gamma \mathrm{e}^{-\frac{t(k_v-\sigma_1)}{2}}\left(k_v+\sigma_1\right)}{l\omega_0^2\left(\mathrm{e}^{-\frac{T(k_v-\sigma_1)}{4}}\sigma_1+\sigma_1\right)} - \frac{\Gamma \mathrm{e}^{-t\left(\frac{k_v}{2}+\frac{\sigma_1}{2}\right)}\left(k_v-\sigma_1\right)}{l\omega_0^2\sigma_1\left(\mathrm{e}^{-\frac{T(k_v+\sigma_1)}{4}}+1\right)}, \tag{A.9}$$

where $\sigma_1 = \sqrt{k_v^2 + 4\omega_0^2}$.

ANNEXES B

Deep RL methods

B.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) [LeCun et al., 1995] are a class of deep learning models specifically designed for processing data with a known grid-like topology, such as images [Goodfellow et al., 2016, LeCun et al., 2015]. CNNs are often used for image classification, segmentation, object detection, and other applications of computer vision. The basic unit of a CNN is the convolutional layer. A convolutional layer is based on the convolution operation borrowed from traditional signal processing. Convolution outputs the aggregated value of its input via a moving window through the signal.

The convolution can be applied both to 1D (ex : time-series) and 2D (ex : image) data. For a 2D input I and a 2D convolutional filter K, convolution is defined as :

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n). \tag{B.1}$$

A CNN layer can be resumed to three steps :

1. *Convolution* : this stage uses a set of learnable filters (kernels) and each filter is passed through the width and the height of the image to produce a 2D feature map (with lesser size then the original input). These maps capture the presence of specific features at different locations in the input.
2. *Activation* : just like in DNNs, the non-linearity is introduced to model the complexity of a system.
3. *Pooling* : A pooling stage replaces the output of activation at a certain location with some statistic of the nearby outputs. Its role is to reduce the dimension of an input. Two main alternatives are average and max pooling. In the average pooling, the output is the mean of an input, while in max pooling, the output is the max of an input window.

CNNs take an input image and apply a series of filters to it. Each filter is a small matrix that is used to extract a specific feature from the image. The output of convolutions is passed through the nonlinear function, like in DNNs, and then the *pooling* stage is applied. There are many variants of CNNs, further details can be found in the book [Goodfellow et al., 2016].

In case of deep RL, CNNs represent a compression tool (representation of an image) that takes an image as input and outputs the latent vector. Initially the weights in CNNs are initialized via some initialization scheme and then trained with back-propagation, just like DNNs.

B.2 Different types of Reinforcement learning

Imitation learning

Imitation learning is a type of reinforcement learning where agents learn the policy by mimicking expert demonstrations. Expert demonstrations usually have high rewards in the given task and imitation learning consists of supervised learning on the state-action pairs of expert transitions. Using imitation, the agent can achieve the reward similar to the expert. Imitation learning can be a simple but effective way to pretrain the agent before the learning, thus helping to converge faster.

Let $\mathcal{D} = \{(s_0, a_0), (s_1, a_1), \ldots\}$ be expert transitions. The role of imitation learning is to train the policy π_θ on $\mathcal{D}$ in a supervised way to obtain similar actions to experts. The 2 most basic and popular methods are Behavior cloning [Pomerleau, 1988] and Dagger [Ross et al., 2011].

Offline Reinforcement learning

Offline RL is similar to Imitation learning, but permits finding the best possible actions from the offline dataset. The difference between the Offline RL and imitation learning, is that Offline RL does not try to mimic the agent, but improve on the given offline trajectories. In certain cases the Offline RL algorithms construct the model from the dataset and train the agent on this model. The good baseline choices for the offline RL method is AWAC [Nair et al., 2020]. Other methods include CQL [Kumar et al., 2020] and IQN.

Model-based Reinforcement learning

Model-based Reinforcement learning (MBRL) is a branch of RL that achieves the objective of RL while modeling internally state transition probability (P) of MDP. First viable algorithms of MBRL include PILCO [Deisenroth and Rasmussen, 2011] which models the dynamics with Gaussian processes. It is a highly sample-efficient model sacrificing computational efficiency of an algorithm. Its main disadvantage is that it can not scale to high-dimensional environments and needs tuning of parameters. Other MBRL algorithm examples include Dreamer [Hafner et al., 2020] that model the dynamics via the *RSSM* (recurrent state space network based on RNN) [Hafner et al., 2019]. Dreamer is a State-of-the-Art MBRL algorithm, being a highly sample-efficient algorithm it requires a lot of numerical computations. In the next paragraph we present the MBPO [Janner et al., 2019] algorithm that has a DNN based internal model of the environment that helps accelerate the training.

MBPO

Model-free offline RL algorithms (DQN, SAC) optimize only control policy, while MBRL algorithms construct the internal model of the environment to increase the sample efficiency of algorithms. MBPO presents an algorithm that uses model-free RL such as SAC as a controller and models the environment dynamics with the predictive model $p_\theta\left(s_t, a_t \rightarrow s_{t+1}\right)$, named ensemble [Dietterich, 2000] of neural networks i.e. several NNs that are trained in parallel. The number of neural networks representing the environment varies among the tasks. The ensemble of NNs permits adding stochasticity to the prediction, because DNN is inherently deterministic and most of the MDPs are stochastic in reality. The algorithm can be resumed to several steps :

1. Collect data from the environment
2. Fit the ensemble model (NNs) to the data
3. Generate simulated data from the model
4. Train model-free RL agent i.e SAC using the simulated data

The steps of the algorithm are explained in Algorithm 6.

Algorithm 6 Model-Based Policy Optimization with Deep Reinforcement Learning

Initialize policy π_ϕ, predictive model p_θ, environment dataset $\mathcal{D}_{\text{env}}$, model dataset $\mathcal{D}_{\text{model}}$
for N epochs do **do**
    Train model p_θ on $\mathcal{D}_{\text{env}}$ via maximum likelihood
    for E steps do **do**
        Take action in the environment according to π_ϕ ; add to $\mathcal{D}_{\text{env}}$
        for M model rollouts **do**
            Sample s_t uniformly from $\mathcal{D}_{\text{env}}$
            Perform k-step model rollout starting from s_t using policy π_ϕ ; add to $\mathcal{D}_{\text{model}}$
        end for
        for G gradient updates **do**
            Update policy parameters on model data : $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi \left(\phi, \mathcal{D}_{\text{model}}\right)$
        end for
    end for
end for

Meta Reinforcement learning

Meta Reinforcement learning is a paradigm known as *learning to learn*. Instead of learning to maximize a particular objective, the algorithm learns how to perform a distribution over tasks $p(\mathcal{T})$. The algorithms of meta RL learn how to optimize the general objective function and learn "faster" for downstream tasks. The most popular method is MAML [Finn et al., 2017] (model agnostic meta learning), which can be applied both for Supervised learning and Reinforcement learning.

B.3 Virtual Environments

Gym

The OpenAI Gym, is a popular framework for developing, comparing and testing RL algorithms. It provides a suite of environments that can test the performance of agents on a wide range of tasks. Custom environments containing user-defined dynamics and animation can be defined. The cart-pole and fish environments can be visualized below.

Figure B.1 – Sample image of cart-pole simulation with openai gym.

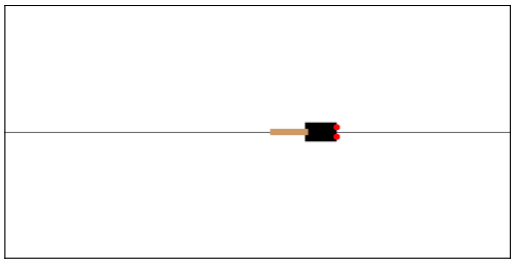

Figure B.2 – Original image used for visual simulation of a robotic fish. The defined dynamics can be found in Chapter 3.

B.4 Robotic fish thrust and speed optimization

PPO hyperparameters for thrust/swim environments

The parameters were tuned with the help of Optuna framework [Akiba et al., 2019]. The complete list of hyperparameters are :

— **Numbers of epochs** K : number of passes through the buffer data that PPO learns from.
— **Batch size** B : number of samples used for the gradient descent update of a neural network. In practice, it should be large enough to avoid biased experience, but not too large to slow the learning. Furthermore, in policy gradient updates, a small batch size may destabilize the learning due to large variance.
— **Learning rate** α : an extent to which we update the neural networks at each step.
— **Discount factor** γ : discount rate of future steps, which tells how the present is more valuable than the future.
— **Network size** N_s : the size of the dense neural network for "action-value" and "policy" function approximations.
— **Number of gradient steps** T : how many gradient steps of learning do we make during every epoch.
— **Entropy coefficient** c_e : the weight of entropy of actions in the total optimized loss. Entropy signifies how random the taken actions are. It permits exploring unknown states.
— **Value coefficient** c_v : the weight of value regression in the total optimized loss.
— **GAE Lambda** : λ coefficient in GAE estimation [see Section 2.3.3.4].

— **Clip range** : ϵ policy clip range for stabilization [see Section 2.3.3.4].
— **Max Grad Norm** : clips the max grad norm of the gradients of the update.
The CNN architecture used in the algorithm is the same as in [Mnih et al., 2015].

Hyperparameter	**thrust**	**target speed**	**speed max**	**visual servoing**
Number of epochs K	40	40	20	40
Batch size (B)	128	128	128	256
Gradient steps (T)	768	768	128	768
discount factor (γ)	0.98	0.98	0.98	0.98
Learning rate (α)	1.7×10^{-5}	1.7×10^{-5}	1.7×10^{-5}	1.7×10^{-4}
Entropy coefficient (c_e)	0.0876	0.003	0.05	0.08765
Clip range	0.1	0.1	0.1	0.1
GAE Lambda	1.0	1.0	1.0	1.0
Max Grad Norm	0.8	0.8	0.8	0.8
Value coefficient (c_v)	0.451	0.451	0.4228	0.4228

Table B.1 – PPO hyperparameters for speed optimization. "thrust" used in simulation and experiment for thrust maximization and speed maximization in simulation. "target speed" is used for reaching a target speed in simulation. "speed maximization" is used for maximizing speed in experiments. The "visual servoing" is used for visual servoing simulation environment.

Simulation parameters for fish thrust/speed maximization

Description	**Value**
Ω	5.8
ξ	1.2
$\Delta\phi$	0.29
ω_0	12.5
Φ	60

Table B.2 – Hyperparameter list for Fish environments simulations.

Description	Value
Ω	5.8
ξ	1.2
C_d	0.254
C_t	12.9e-3
Φ	60

Table B.3 – Hyperparameter list for Fish environments simulations.

Description	Unit	Thrust	Speed
Ω	rad.s^{-1}	5.8	5.8
ξ	-	1.2	1.2
$\Delta\phi$	rad	0.29	-
ω_0	rad.s^{-1}	12.5	-
C_d	-	-	0.254
C_t	-	-	12.9e-3
Φ	rad	0.31	0.31

Table B.4 – Physical parameters list used in fish environment simulations.

Layer Index	Encoder	Decoder
0	Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2))	ConvTranspose2d(32, 32, kernel_size=(1, 1), stride=(1, 1))
1	ReLU()	ReLU()
2	Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1))	ConvTranspose2d(32, 32, kernel_size=(1, 1), stride=(1, 1))
3	ReLU()	ReLU()
4	Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1))	ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
5	ReLU()	ReLU()
6	Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))	ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
7	ReLU()	ReLU()
8	Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))	ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
9	ReLU()	ReLU()
10	Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))	ConvTranspose2d(32, 1, kernel_size=(4, 4), stride=(2, 2))
11	ReLU()	Sigmoid()
12	Flatten(start_dim=1, end_dim=-1)	–
–	Linear(in_features=39200, out_features=256, bias=True)	Linear(in_features=6, out_features=256, bias=True) (fc0)
–	Linear(in_features=256, out_features=6, bias=True) (fc_mu)	Linear(in_features=256, out_features=39200, bias=True) (fc)
–	Linear(in_features=256, out_features=6, bias=True) (fc_logvar)	–

Table B.5 – VAE architecture used for visual-servoing in simulation, optimized in performance and memory for this task. First input image (84, 84, 1) passes through 2d CNN layers with linear layers at the end. Last linear layers encode the mean and standard deviation of the Gaussian distribution in the latent space. A sample is then drawn from this Gaussian distribution and fed into another set of linear layers that incrementally increase the dimensionality of the data. Subsequently, 2D transposed convolutional layers (layer index : 0-10) are applied to upscale the feature maps, ultimately reconstructing an output that matches the size of the original input image.

Learning to achieve a fixed target speed (virtual environment) : DROQ

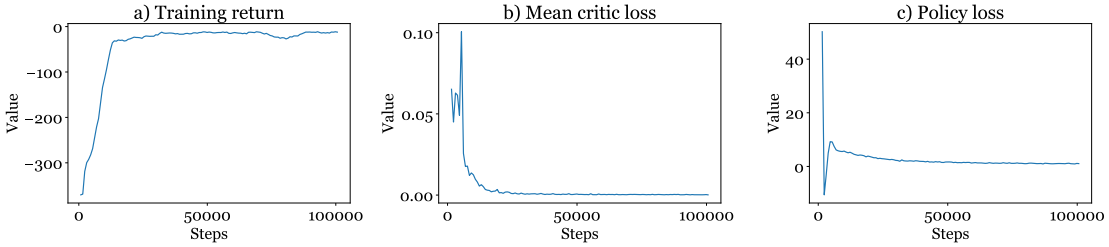

Figure B.3 – Learning statistics for achieving a fixed target speed with DROQ. a) Training reward recorded after every episode. b) Critic ($Q(s,a)$) loss evolution through time. c) Policy loss evolution through time.

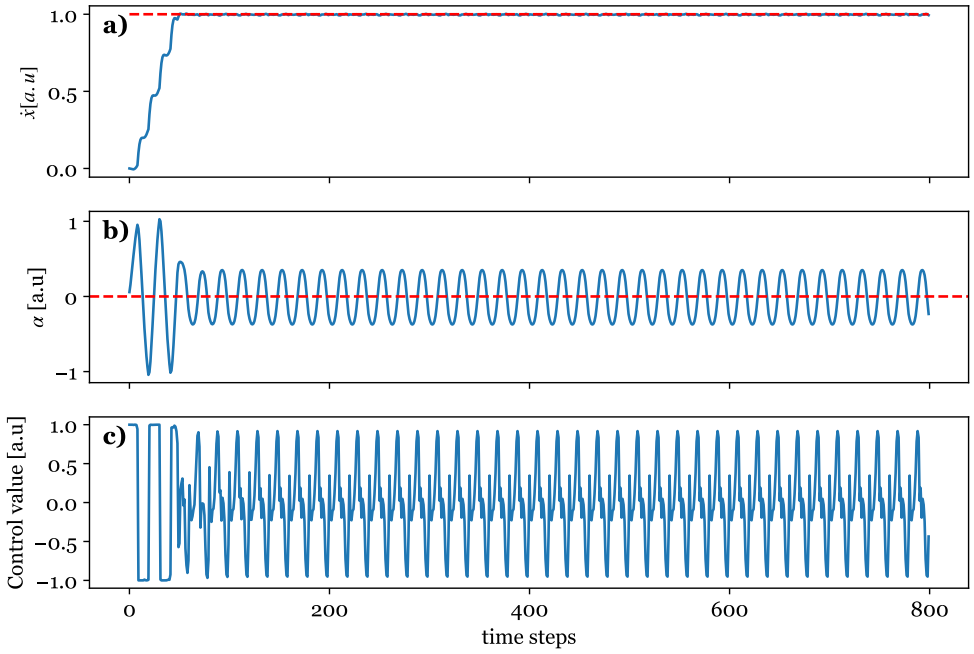

Figure B.4 – Inference results with DROQ at the end of 10^5 learning time steps. a) Speed $\dot{x}$ evolution superposed with target speed (red dashed line). b) Fin angle evolution through time. First 50 time steps are related to the high amplitude undulations. c) Action evolution through time. High amplitude control is followed by medium amplitude control.

Description	**Value**
number of training steps	100000
batch size	256
learning rate	0.0003
hidden units	[256, 256]
replay buffer size	1000000.0
discount (γ)	0.99
tau	0.005
entropy tuning	True
entropy coef	0.2
critic updates per step	20
eval episodes interval	50
gradients steps of update	128
start steps (warm-up)	500
target update interval	1
layer normalization	1
target entropy	-1.0
target dropout rate	0.005
number of critics	2

Table B.6 – Hyperparameter list for DROQ learning. We adapted the code from [Hiraoka et al., 2021] and trained the model at the end of every episode.

ANNEXES C

Servomotor model

We construct the servo-motor simulation model based on the electrodynamics equations. We consider servomotor based on the Direct Current (DC) motor, following the technical documentation of the servo used [1].

The basic servo-motor has three components :

1. *DC motor with reductor*. It converts direct current electrical energy into mechanical energy. It is based on the principle that the conductor carrying the current in the magnetic field is subject to the "Lorentz" force. The rotational speed of DC motor is in direct relation with the applied voltage and the load mounted on the output shaft of a motor. We consider the "Permanently excited brushed DC motor" (DC motor with permanent magnets in the stator) vizualised in Fig. C.1. The motor is simplified to the the resistance R_a, inductance L_a, rotating shaft M and voltage E_a applied to motor terminals.

   Electrical model of a DC motor is well known and is described in Eq. (C.2), where $E_{emf}(t) = K_b\dot{\theta}_M(t)$ is a back electromotive force opposing the movement, which is proportional to the rotational speed of a motor.

   A rotating shaft usually have a high speed and low torque and for this reason the mechanical gearbox is coupled to the motor to increase the torque and decrease the shaft rotation speed with the factor of $\left(\frac{N2}{N1}\right)$, where $\frac{N2}{N1}$ is a reduction ratio of a gearbox to the motor. Redactor also influences the total inertia of rotation. It is described in Eq. (C.3), where the total inertia of the system is the sum of motor inertia and load inertia on a reduced speed shaft.

   Electromechanical torque balance of a system is established in Eq. (C.1). The signification of the terms is defined as :

   (a) $J_{total}\ddot{\theta}_M(t)$ - a torque needed to rotate the inertia with acceleration $\ddot{\theta}$

   (b) $b\dot{\theta}_M(t)$ - a viscous torque

   (c) $T_M(t)$ - a torque produced by a motor

   (d) $T_L(t)$ - a load torque with damping d_1. For simplicity, we neglect the damping of the load for the sake of this derivation.

2. *Power driver*. It has a set of smart switches (transistors) that convert the given voltage of 6V to the desired Voltage via Pulse Width Modulation (PWM) of a given voltage (6V $\rightarrow$ [0,6] V).

3. *Feedback and controller*. All servomotors have a feedback on the angular position θ. Some of the angular feedback devices are based on potentiometers or hall-effect sensors. The role

1. https://www.hiteccs.com/public/uploads/data_sheet/HS-5086WP_DataSheet-1642816390.pdf

of a controller is to minimize the error between the desired and actual angular positions such that the servo-motor goes to the precise location. As discussed in Section 2.1.1, PID is the most popular controller and we suppose that control of a servomotor is defined by a classical PID on the angular position.

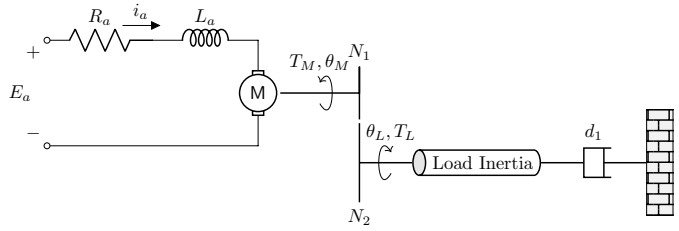

Figure C.1 – schema-block of a DC motor with the load mounted on the redactor shaft. The motor producing torque $T_M(t)$, consists of the voltage source E_a, the resistance R_a and inductance L_a in sequence. The rotor shaft M is coupled to the load torque T_L via mechanical reduction (N_2/N_1 - gear ratio) that decreases the speed of rotation while increasing the applied torque.

Before understanding the inner workings of a servomotor, we need to know the details of DC motor functioning. The dynamics of DC motor (Fig. C.1) without damping $d1$ is described by the following set of equations :

$$J_{total}\ddot{\theta}_M(t) + T_{visc}(t) = T_M(t) - T_L(t)\left(\frac{N1}{N2}\right) \tag{C.1}$$

$$L_a\frac{di_a(t)}{dt} + R_a i_a(t) = E_a - E_{emf}(t) \tag{C.2}$$

$$J_{total} = J_{motor} + \left(\frac{N1}{N2}\right)^2 J_{load}, \tag{C.3}$$

where the mechanical balance i.e Eq. (C.1) is derived using Newton's laws and the balance of electrical potentials in Eq. (C.2) is obtained using Kirchhoff's second law. The Eq. (C.3) defines total inertia of the system.

The motor torque ($\mathbf{T_M}$) has linear dependence with armature current [see Eq. (C.4)]. Also, back-electromotive force ($\mathbf{E_{emf}}$) and viscous friction torque ($\mathbf{T_{visc}}$) is linearly proportional to rotational speed of the motor [see Eqs. (C.5) and (C.6)] :

$$T_M(t) = K_t i_a(t) \tag{C.4}$$

$$E_{emf}(t) = K_b\dot{\theta}_M(t) \tag{C.5}$$

$$T_{visc}(t) = b\dot{\theta}_M(t), \tag{C.6}$$

where K_t, K_b and b are constants.

The power driver component of the servo motor is responsible for regulating the applied (PWM) voltage $\mathbf{E_a}$ on the DC motor from the 6V voltage source. The control is governed by the following PID equations :

$$E_a(t) = K_p e(t) + K_i\int e(t)dt + K_d\dot{e}(t) \tag{C.7}$$

$$e(t) = \theta_{Ref}(t) - \theta_L(t) \tag{C.8}$$

Applied voltage $E_a(t)$ on a servomotor features proportional, integral and derivative terms with respect to error in position. $E_a(t)$ is limited by upper and lower bounds of 6V which corresponds to the voltage of the power source.
Servomotor is a brushed DC motor with the gear, power (PWM) driver and potentiometer to control it precisely on position [see Fig. C.2].

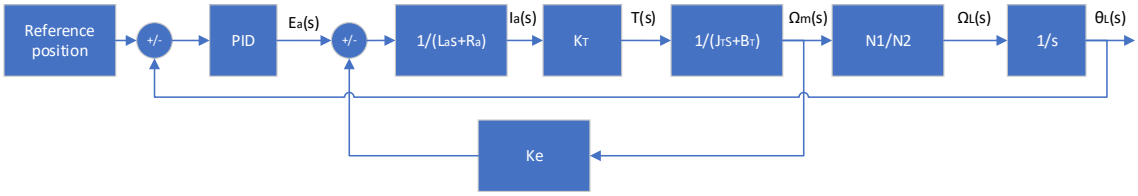

Figure C.2 – schema-block of servomotor. s corresponds to the derivative.

Neglecting inertia, inductance as well as the load torque $T_L = 0$ and assuming the P component of PID control is dominant, from Eqs. (C.1) and (C.2) we obtain the following ODEs :

$$i_a(t) = (E_a - K_b\dot{\theta}_M)/R \tag{C.9}$$

$$\dot{\theta}_M(t) = \frac{K_t}{R.(b + K_b/R)} sat(K_p(\theta_{Ref}(t) - \theta_L(t)), -E_{source}, E_{source}) \tag{C.10}$$

Due to the nature of electrical components in the power drive and integral/derivative terms of PID controller, the saturation function is smooth and thus Eq. (C.10) closely resembles discussed before Eq. (3.2). Specifically, the rotational speed of a load shaft can be defined as :

$$\dot{\theta}_L(t) = \frac{K_t}{R.(b + K_b/R)} sat(K_p(\theta_{Ref}(t) - \theta_L(t)), -E_{source}, E_{source}) \approx \tag{C.11}$$

$$\Omega \tanh\left(\frac{1}{\Delta}(\theta_{Ref}(t) - \theta_L(t))\right) \tag{C.12}$$

Servo-motor power We now aim to establish the power balance. Multiplying Eq. (C.2) by the current value $i(t)$ gives electrical power balance and multiplying Eq. (C.1) by the angular speed $\dot{\theta}_M(t)$.

$$\underbrace{K_t i_a(t)\dot{\theta}_M(t)}_{\text{Total mechanical power}} = \underbrace{J_{total}\ddot{\theta}_M(t)\dot{\theta}_M(t)}_{\text{rotation}} + \underbrace{b\dot{\theta}_M^2(t)}_{\text{friction}} + \underbrace{T_L(t)\dot{\theta}_L(t)}_{\text{output}} \tag{C.13}$$

$$\underbrace{E_a i_a(t)}_{\text{Total electrical power}} = \underbrace{L\frac{di_a(t)}{dt}i_a(t)}_{\text{armature inductunce}} + \underbrace{Ri_a^2(t)}_{\text{Joule loss}} + \underbrace{K_b\dot{\theta}_M(t)i_a(t)}_{\text{Transmitted mechanical power}} \tag{C.14}$$

Equations (C.13) and (C.14) are the power balance of the DC motor and do not capture the power loss due to electronics commutation etc. The total power balance :

$$P_{elec} = P_{meca} + P_{loss} = P_{meca} + P_{loss-const} + I_a * C_L + I_a^2.R \tag{C.15}$$

Where

— P_{meca} is a mechanical power
— $P_{loss-const}$ is a power loss due to commutation electronics etc.
— $I_a * C_L$ ia a linear loss term due to the voltage drop at the brushes of dc motor
— $I_a^2 * R$ ia a Joule heat dissipation through total resistance of the circuit

ANNEXES D

Learning to Swim

D.1 Experimental Setup for Robotic Fish swimming

D.1.1 Water tunnel

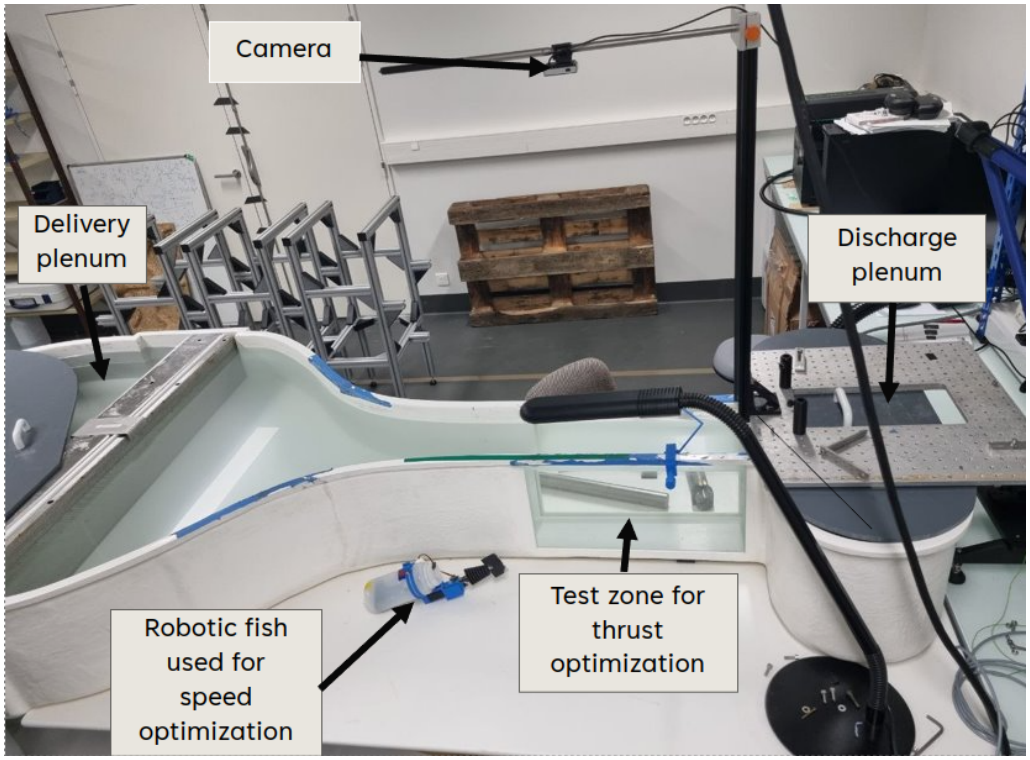

Figure D.1 – Water tunnel used in the experiences with the robotic fish.

All the experiments have been conducted in the water tunnel of Rolling Hills Research Corporation, model 0710. This tunnel in Appendix D.1.1 measures 270 cm in length, 110 cm in width and 30 cm in height with an approximate maximum volume of 500L. We used the desalinated water of the density $\rho = 1000\ \mathrm{kg} \cdot \mathrm{m}^{-3}$ and kinematic viscosity, $\nu = 10^{-6}\ \mathrm{m}^2 \cdot \mathrm{s}^{-1}$. The water tunnel comprises three sections :

1. *Delivery Plenum* : This is the initial section where the water flow starts its journey. The water is pumped with an asynchronous motor (1.5 kW, 400 V, 50 Hz, 6 poles) that is driven by the frequency command ; the speed of the flow is calibrated in the next section.

The water channeled into the tank undergoes multiple filters to ensure laminar flow within the test section.

2. *Test Section* : This is where the robotic fish is situated for experiments. Its easy accessibility and clear visibility make it especially suitable for experimental procedures with the static robotic fish. Moreover, it's in this section that we conducted the experiments with the moving fish.
3. *Discharge Plenum* : This section serves to extract the water, which is then recirculated back to the pump via a closed-loop system, perpetuating the cycle.

D.1.2 Water tunnel speed calibration

Water tunnel speed can be fitted with a linear curve with the coefficient of value 0.00223.

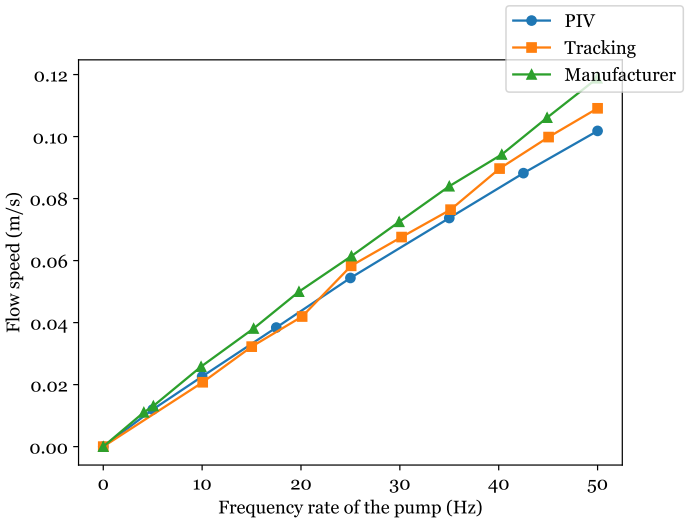

Figure D.2 – Calibration of the water pump. Three indicators of determining the flow speed as the function of the command frequency. *PIV* (Particle Image Velocimetry) corresponds to the measurements of the speed of small particles in the laminar flow. *Tracking* corresponds to the measurement of speed of the colored particles in the flow via the camera mounted above the water tank. *Manufacturer* corresponds to the speed specified by the manufacturer. The small gap between *PIV* and *Tracking* methods can be explained by the different levels of water for the test cases.

D.1.3 Drag force determination

The procedure consists of varying the speed of the water tunnel and recording the drag force of the stationary robotic fish attached to the force sensor.

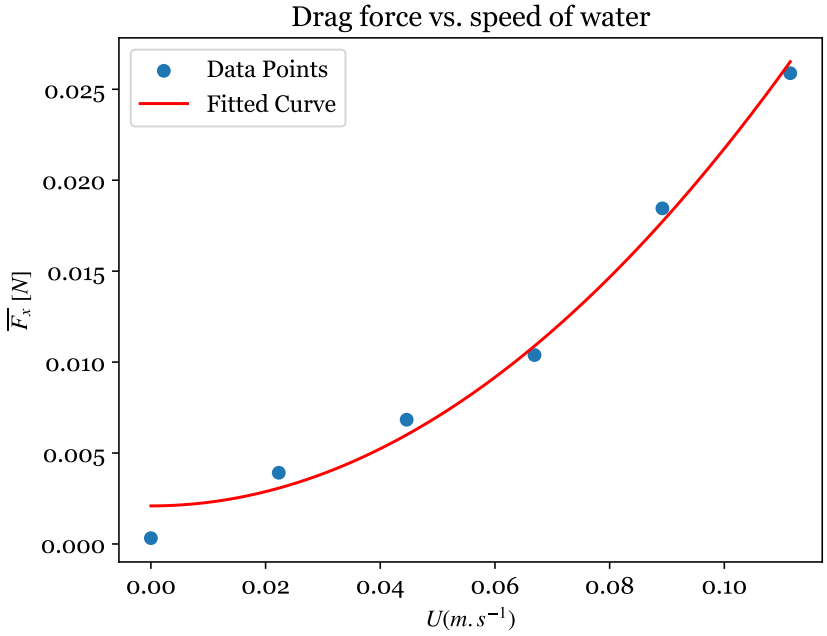

Figure D.3 – Drag force as a function of the speed of a robotic fish. Assuming that the pressure drag is $\overline{F_x} = \rho C_d L^2 U^2$, the best interpolation gives $C_d = 0.2$ with $\rho = 1000\,\mathrm{kg}\cdot\mathrm{m}^{-3}$ and $L = 10\,\mathrm{cm}$.

ANNEXES E

Thrust force optimization of a robotic fish

E.1 Fluid-Structure Interaction Simulation

To simulate the fish swimming driven by an optimal command, we used the software COMSOL 6.1 following the approach outlined in [Curatolo and Teresi, 2015]. The 2D computational domain covers a rectangle with dimensions 100×20 cm^2, representing water. In Fig. E.1a, we show the geometry of the simulation. The two horizontal borders represent slip boundary conditions for the velocity, while the left and right borders are associated with entrance and exit boundary conditions. The swimmer is approximated by a viscoelastic beam of Young Modulus 10^4 Pa, a Poisson ratio 0.3 and a viscosity 10^{-4} Pa.s. It navigates toward the left. The thickness $t(X)$ of the viscoelastic beam is given by the relation

$$t(X) = H\frac{X}{L}\left(1 - \frac{X}{L}\right)e^{-\frac{X}{L}},$$

where X is the curvilinear distance along the midline, measured from the head. Here $H = 4$ cm and $L = 10$ cm, such that the thickest part of the beam measures 1.3 cm, see Fig. E.1b.
To model the antagonistic muscle action, we impose on the swimmer that the equilibrium component ϵ_{xx} of the strain varies spatiotemporally :

$$\epsilon_{XX}(X, Y, t) = 0.01(X/L)^2(Y/H)a(t),$$

where $a(t)$ drives the motion dynamics, and ϵ is the strain tensor [Landau et al., 1986]. This forcing modifies the equilibrium length of each part of the body in an opposite manner : when the superior part of the swimmer elongates its length, the other part contracts it, following the dynamics of $a(t)$. $a(t)$ can be a wave function or defined as $a(t) = \mathrm{sign}(v(t))$, where $v(t)$ is the normal velocity of the swimmer at the tail. The numerical value is chosen such that the typical amplitude of the tail oscillation is close to 0.2.
The fluid-structure problem is solved using a fully coupled approach and the PARDISO linear solver ; the nonlinear problem is tackled with a Newton algorithm. Because the swimmer deforms its shape, the mesh is adapted using a Yeoh method. To avoid excessively large deformations in the mesh due to the swimmer's movement, the entire computational domain is remeshed is the discretisation is too distorted. Approximately 7,000 vertices are needed for almost 40,000

degrees of freedom to coarsely solve the complete fluid-structure interaction on the whole domain, and predict a correct swimming velocity. A finer element distribution is ensured at the head and tail (Fig. E.1d.) However, to accurately capture the wake, 300,000 elements are required, as shown in Fig. E.1d. The typical time step used is 10^{-3} s. The center of mass of the swimmer is computed at each time step during the simulation. This comprehensive setup enables us to study and analyze the swimming behavior of the robotic fish driven by the optimal command obtained through the reinforcement learning process.

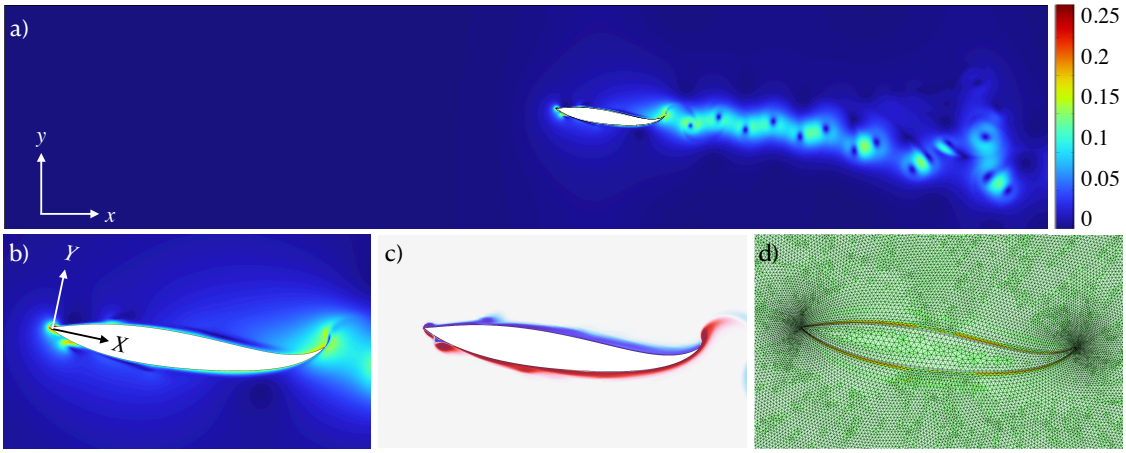

Figure E.1 – a) Setup of the computational domain. The white region represents the swimmer body, and the color codes the speed in m/s. b) Zoom around the swimmer body. c) Typical vorticity field around the swimmer. d) Zoom around the swimmer to show the typical mesh precision used for the simulations.

E.2 Optimal Bang-bang controller for thrust maximization

Following the Pontryagin principle, we aim to determine the best command to optimize the dimensionless force, averaged for a duration T :

$$I = \frac{\overline{F_x}}{K\Lambda^2\omega_0^2} = \frac{1}{\Lambda^2\omega_0^2}\frac{1}{T}\int_0^T \dot{\alpha}^2 dt, \quad \Lambda = \lambda\Phi \tag{E.1}$$

under the dynamics :

$$\dot{\alpha} = \omega \tag{E.2}$$

$$\dot{\omega} = -\xi\omega_0\omega - \omega_0^2(\alpha - \alpha_c),\ |\alpha_c| \leq \Lambda, \quad \alpha_c = \lambda\phi \tag{E.3}$$

$$\dot{\phi} = \Omega \tanh\frac{\phi_c - \phi}{\Delta}. \tag{E.4}$$

Within the variational methods framework, we write the Lagrangian :

$$L = \omega^2 + p_1(\dot{\alpha} - \omega) + p_2(\dot{\omega} + \xi\omega_0\omega + \omega_0^2(\alpha - \lambda\phi)) + p_3\left(\dot{\phi} - \Omega\tanh\frac{\phi_c - \phi}{\Delta}\right), \tag{E.5}$$

where we have introduced three Lagrangian multipliers, $p_{i=1..3}$ to impose the dynamics of α, ω and ϕ. We reformulate this problem using the natural Hamiltonian :

$$H = -L + p_1\dot{\alpha} + p_2\dot{\omega} + p_3\dot{\phi} \tag{E.6}$$

$$H = -\omega^2 + p_1\omega + p_2\left(-\xi\omega_0\omega - \omega_0^2(\alpha - \lambda\phi)\right) + p_3\Omega\tanh\frac{\phi_c - \phi}{\Delta}. \tag{E.7}$$

The co-state equations for this system are :

$$\dot{p}_1 = -\partial_\alpha H = \omega_0^2 p_2 \tag{E.8}$$

$$\dot{p}_2 = -\partial_\omega H = 2\omega - p_1 + \xi\omega_0 p_2 \tag{E.9}$$

$$\dot{p}_3 = -\partial_\phi H = -\omega_0^2\lambda p_2 + p_3\frac{\Omega}{\Delta}\cosh^{-2}\frac{\phi_c - \phi}{\Delta} \tag{E.10}$$

Following the Pontryagin seminal idea, the value of the control parameter ϕ_c can be chosen in order to maximize the Hamiltonian value. The bang-bang controller will be optimal if the fourth term in the Hamiltonian (E.7) which contains ϕ_c is linear, which is not the case here. Nevertheless, a square forcing for $\phi_c = \pm\Phi$ will maximize the Hamiltonian (E.7) : if $p_3 > 0$, we choose $\phi_c = \Phi$ and $\phi_c - \phi$ will be positive. On the contrary, if $p_3 < 0$, we choose $\phi_c = -\Phi$ and $\phi_c - \phi$ will be negative :

$$\phi_c = \Phi\ \mathrm{sign}(p_3), \tag{E.11}$$

and we recover a bang-bang controller.

E.2.1 Computations for fast servomotors

In the limit of a fast servomotor, the variable ϕ and therefore α_c follow adiabatically ϕ_c. Consequently, we set that the variable $\alpha_c \in [-\Lambda, \Lambda]$ is the control variable. In this limit, the dynamics for the fin angle α is described with :

$$\dot{\alpha} = \omega \tag{E.12}$$

$$\dot{\omega} = -\xi\omega_0\omega - \omega_0^2(\alpha - \alpha_c),\ |\alpha_c| \leq \Lambda \tag{E.13}$$

While the Hamiltonian H from Eq. (E.7) reduces to

$$H = -\omega^2 + p_1\omega + p_2\left(-\xi\omega_0\omega - \omega_0^2(\alpha - \alpha_c)\right). \tag{E.14}$$

We remark that $H = \alpha_c(\omega_0^2 p_2) + ...$ Hence, by setting $\alpha_c = -\Lambda$, if $p_2 > 0$ and $\alpha_c = \Lambda$, if $p_2 < 0$, we impose that the Hamiltonian is always minimum. Consequently, if p_2 oscillates then the control α_c also oscillates between the two constant values extreme values $\pm\Lambda$. The optimal command is therefore :

$$\alpha_c = \Lambda\mathrm{sign}(p_2), \tag{E.15}$$

which corresponds to the classical Bang-Bang controller. We now study the dynamics of the $p_{1,2}$. The equations of the co-states are :

$$\dot{p}_1 = -\partial_\alpha H = p_2\omega_0^2 \tag{E.16}$$

$$\dot{p}_2 = -\partial_\omega H = 2\omega - p_1 + \xi\omega_0 p_2. \tag{E.17}$$

The equation for p_2 is written in the form :

$$\ddot{p}_2 - \xi\omega_0\dot{p}_2 + \omega_0^2 p_2 = 2\dot{\omega}, \;\; p_2(0) = 0, \;\; p_2(T) = 0 \tag{E.18}$$

which represents a linear oscillator forced by $\dot{\omega}$ and submitted to an injection of energy (the term proportional to $\dot{p}_2$). As the equation for α of those of a harmonic oscillator if α_c is constant, the r.h.s. of eq. (E.18) is harmonic. Consequently, the co-state p_2 should be a harmonic function whose amplitude grows in time because of the injection of energy. This demonstrates that p_2 oscillates, yielding oscillations in α_c.

In the next part of this section we study the system (E.2,E.3). We impose that $\alpha_c = \Lambda$ for $t \in [0, T/2]$ and $\alpha_c = -\Lambda$ for $t \in [T/2, T]$.

We solve the equation for the angle α in the first half period :

$$\alpha = \Lambda + e^{-\frac{\xi\omega_0}{2}t}\left(a\cos\nu t + b\sin\nu t\right) \tag{E.19}$$

$$\nu = \frac{1}{2}\omega_0\sqrt{4-\xi^2}, \tag{E.20}$$

where a and b are fixed by two boundary conditions. Applying the condition of continuity and differentiability :

$$\alpha(0) = -\alpha(T/2) \tag{E.21}$$

$$\dot{\alpha}(0) = -\dot{\alpha}(T/2), \tag{E.22}$$

the unknowns a and b are determined :

$$a = \frac{\Lambda\xi\omega_0\sin\left(\frac{T\nu}{2}\right) - 2\Lambda\nu\cos\left(\frac{T\nu}{2}\right) - 2\Lambda\nu e^{T\xi\omega_0/4}}{2\nu\left(\cosh\left(\frac{1}{4}\xi T\omega_0\right) + \cos\left(\frac{T\nu}{2}\right)\right)} \tag{E.23}$$

$$b = -\frac{\Lambda\xi\omega_0\cos\left(\frac{T\nu}{2}\right) + 2\Lambda\nu\sin\left(\frac{T\nu}{2}\right) + 2\Lambda\nu e^{T\xi\omega_0/4}}{2\nu\left(\cosh\left(\frac{1}{4}\xi T\omega_0\right) + \cos\left(\frac{T\nu}{2}\right)\right)}. \tag{E.24}$$

We remark here that the above formulation is not sensitive to the sign of $\xi - 2$, because the trigonometric functions become hyperbolic if their argument is imaginary. It remains to compute the dynamics for p_2. In fact, the Eq. (E.18) can be solved analytically, with the boundary conditions

$$p_2(0) = 0 \tag{E.25}$$

$$p_2(T/2) = 0 \tag{E.26}$$

$$\dot{p}_2(0) = -\dot{p}_2(T/2). \tag{E.27}$$

It appears that the last condition for the differentiability of p_2 is automatically satisfied if p_2 is null at $t = 0$ and $t = T/2$, because in Eq. (E.18), the forcing ω is continuous.

We show in Fig. E.2, various temporal evolution of α and p_2 in the interval $[0, T]$, for various values of the damping parameter ξ.

Consequently, the system provides oscillatory solutions independently of the chosen T. We remark in the Fig. E.2 that p_2 changes its sign as $\dot{\alpha}(t)$ does, such that a rule of thumb is to induce the change of $\dot{\alpha}_c$ as $\dot{\alpha}(t)$ ...is small enough, e.g. of order $0.1\Phi\omega_0$.

The average thrust writes :

$$I = -\frac{1}{\Lambda^2\omega_0^2}\frac{2}{T/2}\int_0^{T/2}\alpha\ddot{\alpha}dt \tag{E.28}$$

$$= \frac{4}{\xi\omega_0 T}\frac{2\sinh\left(\frac{1}{4}\xi T\omega_0\right) - \xi\omega_0/\nu\sin\left(\frac{T\nu}{2}\right)}{\cos\left(\frac{T\nu}{2}\right) + \cosh\left(\frac{1}{4}\xi T\omega_0\right)}. \tag{E.29}$$

The function I presents a maximum near $T = \frac{2\pi}{\omega_0}$, as shown in Fig. E.3.
This dimensionless thrust can be further optimized by computing the value T^* that renders it maximal. We remark, at this point, that the parameter T always appears multiplied by ω_0 in Eq. E.29, since ν is proportional to ω_0 (as defined in Eq. E.20) : I only depends on ξ and $\omega_0 T$.
In Fig. E.4, we plot $\omega_0 T^*$ and $I^\star$ as a function of ξ, which corresponds to the optimal thrust for fast servomotors.

E.2.2 Limit of small damping : $\xi \to 0$

In fact, assuming a small value for ξ, we show that the thrust is maximized at $T = T^\star$ defined by :

$$T^\star = \frac{2\pi}{\omega_0}\left(1 + \xi^4\frac{1}{384}\left(12 - \pi^2\right)\right) + o(\xi^6), \tag{E.30}$$

while the optimal thrust $I^\star$ writes :

$$I^\star = \frac{16}{\pi^2\xi^2} + \frac{\pi^2 - 9}{3\pi^2} + o(\xi^2). \tag{E.31}$$

In the Fig. E.4, we show the influence of the damping factor ξ on the optimal period and thrust.
In the limit of small damping, $\xi \ll 1$, the optimal thrust diverges. The reason is that in this limit the oscillator in α resonates with the forcing α_c, since α_c change its sign on the period $2\pi/\omega_0$.
This is seen by taking the limit $\xi \to 0$ for the expressions a and b :

$$a = -\Lambda - \frac{\xi\left(\Lambda\left(T\omega_0 - 2\sin\left(\frac{T\omega_0}{2}\right)\right)\right)}{4\left(\cos\left(\frac{T\omega_0}{2}\right) + 1\right)} + O\left(\xi^2\right) \tag{E.32}$$

$$b = -\frac{\Lambda\sin\left(\frac{T\omega_0}{2}\right)}{\cos\left(\frac{T\omega_0}{2}\right) + 1} - \frac{\xi\Lambda}{2} + O\left(\xi^2\right), \tag{E.33}$$

The constant a diverges as $T \to \frac{2\pi}{\omega_0}$, such that the swimming amplitude also diverges, and so the thrust.
In this limit we find that :

$$\alpha(0) = -\frac{4\Lambda}{\pi\xi} - \frac{\xi\left(\left(\pi^2 - 9\right)\Lambda\right)}{12\pi} + O\left(\xi^3\right) \tag{E.34}$$

$$\dot{\alpha}(0) = -\frac{\Lambda\omega_0}{\pi} + \frac{\xi^2\Lambda\omega_0}{8\pi} + O\left(\xi^3\right) \tag{E.35}$$

E.2.3 Limit of large damping : $\xi \to \infty$

In this limit, we get the following relation for the thrust :

$$I = \frac{2}{\xi^2}\left(1 - \frac{4}{\xi T\omega_0} + \frac{1}{\xi^2} - \frac{T^2\omega_0^2}{48\xi^2}\right)$$

The maximum of the thrust is obtained by differentiating the above formula with respect to T and computing $T^\star$ that zeroes this derivative. For large ξ, we obtain the optimal period $T^\star$ and thrust $I^\star$:

$$T^\star = \frac{1}{\omega_0}(96\xi)^{1/3} + o(\xi^{1/3}) \tag{E.36}$$

$$I^\star = \frac{2}{\xi^2}\left(1 - \frac{\left(\frac{3}{2}\right)^{2/3}}{\xi^{4/3}}\right) \tag{E.37}$$

In this limit, the system is over-damped. A boundary layer appears near $t = 0$, the size of this boundary layer is $1/(\xi\omega_0)$. This value is deduced by balancing the second derivative term with the damping term. Following the typical techniques for the asymptotic limit, we get

$$\ddot{\alpha}_i + \xi\omega_0\dot{\alpha}_i = 0 \tag{E.38}$$

$$\xi\omega_0\dot{\alpha}_o + \omega_0^2(\alpha_o - \Lambda) = 0 \tag{E.39}$$

$$\alpha(t) = \alpha_o(t) + \alpha_i(t) - \alpha_o(0), \tag{E.40}$$

where $\alpha_i(t)$ is the inner approximation of $\alpha(t)$ near $t = 0$, i.e. the inner region (where the function is rapidly varying). $\alpha_o(t)$ the outer region, where the function is slowly varying. We find that

$$\alpha(0) = -\frac{\sqrt[3]{\frac{3}{2}}}{\xi^{2/3}}\Lambda \tag{E.41}$$

$$\dot{\alpha}(0) = -\frac{\Lambda\omega_0}{\xi} \tag{E.42}$$

E.2.4 Analysis of the swinging strategy

Here, we would like to measure the efficiency of the swinging strategy at least for $C = 0$, where C is defined in the main text. It consists in changing the sign of α_c as $\dot{\alpha}(t)$ zeroes. In the cruising regime, we have computed so far, this strategy predicts that $\dot{\alpha}(0) = \dot{\alpha}(T/2)$ should be zero. We therefore compute $\dot{\alpha}(0)$ to test the strategy :

$$\dot{\alpha}(0) = -\frac{2\Lambda\omega_0 \sin\left(\frac{1}{4}\sqrt{4-\xi^2}T\omega_0\right)}{\sqrt{4-\xi^2}\left(\cos\left(\frac{1}{4}\sqrt{4-\xi^2}T\omega_0\right) + \cosh\left(\frac{1}{4}\xi T\omega_0\right)\right)}.$$

This expression predicts that $\dot{\alpha}(0) = 0$ for $T = T_s$ and the thrust I_s :

$$T_s = \frac{4\pi}{\sqrt{4-\xi^2}\omega_0} \tag{E.43}$$

$$I_s = \frac{2\sqrt{4-\xi^2}}{\pi\xi}\coth\left(\frac{\pi\xi}{2\sqrt{4-\xi^2}}\right) \tag{E.44}$$

In Figure (E.5), we compare the optimal period $T^\star$ with T_s, as well as the resulting thrusts I : It appears that the swinging strategy is very efficient in automatically choosing the optimal period for relatively low damping ξ. Nevertheless, we remark that the resulting thrust obtained with $T = T_s$ is very close to the optimal one for $\xi < 1.5$. Hence, the swinging strategy appears to be very efficient in reaching the optimal thrust without knowing the values of the physical parameters ω_0 and ξ.

E.2.5 Computations for slow servomotors

Here we detail the computations obtained for servomotor which can not follow the command and work at maximal angular velocity Ω. In this limit, the Eq. (E.4) becomes $\dot{\phi} = \pm\Omega$. If we assume a periodic solution, during the first half period, we deduce :

$$\phi(t) = -\frac{\Omega T}{4} + \Omega t. \tag{E.45}$$

The integration constant has been determined by assuming $\phi(0) = -\phi(T/2)$. We then solve the equation for α :

$$\alpha = \lambda\frac{-\xi\Omega - \Phi\omega_0 + t\omega_0\Omega}{\omega_0} + c_1 e^{\frac{1}{2}t\left(-\sqrt{\xi^2-4}\omega_0 - \xi\omega_0\right)} + c_2 e^{\frac{1}{2}t\left(\sqrt{\xi^2-4}\omega_0 - \xi\omega_0\right)}, \tag{E.46}$$

where c_1 and c_2 are defined through the boundary conditions (E.21,E.22). We otbain that the dimensionless thrust writes :

$$I = \frac{2\Omega^2\left(-A(\xi^2-1)\Omega E + \xi(\xi^2-3)\Omega F + \xi AB(D+C)\right)}{\xi AB^3\omega_0^3(D+C)}, \tag{E.47}$$

where

— $A = \sqrt{\xi^2-4}$
— $B = \frac{\Phi\omega_0}{\Omega}$
— $C = \cosh(B)$
— $D = \cosh(AB)$
— $E = \sinh(B)$
— $F = \sinh(AB)$

E.2.6 Limit of small damping

By taking the limit $\xi \to 0$ on (E.47), we deduce :

$$I = \frac{4\Phi^2\omega_0^2}{15\Omega^2}, \tag{E.48}$$

such that $F_x = 2K\lambda^2\Omega^2$, as claimed in the main text.

E.2.7 Limit of large damping

By taking the limit $\xi \to \infty$ on (E.47), we deduce :

$$I = \frac{2}{3\xi^2} \tag{E.49}$$

Why the swinging strategy yields a resonance ?

In the main text, we have hypothesized that an efficient way to drive the swimmer to its resonnance without the knowledge of the frequency modes of deformations. The idea is to impose that

$$\alpha_c = \Lambda \text{sign}(\dot{\alpha}). \tag{E.50}$$

With this command, the damped oscillator can be solved in each half plane of the phase portrait $(\alpha, \dot{\alpha})$:

$$\ddot{\alpha} + \xi\dot{\alpha} + \omega_0^2(\alpha - \Lambda) = 0, \quad \dot{\alpha} < 0 \tag{E.51}$$

$$\ddot{\alpha} + \xi\dot{\alpha} + \omega_0^2(\alpha + \Lambda) = 0, \quad \dot{\alpha} > 0. \tag{E.52}$$

Assuming a periodic motion for which $\dot{\alpha} > 0$ for $0 < t < T/2$ and $\dot{\alpha} < 0$ for $T/2 < t < T$, the motion on the second half period is deduced from the the value of α in $0 < t < T/2$:

$$\alpha = \Lambda + e^{-\frac{\xi\omega_0}{2}t}\left(a\cos\nu t + b\sin\nu t\right) \tag{E.53}$$

$$\nu = \frac{1}{2}\omega_0\sqrt{4 - \xi^2}, \tag{E.54}$$

where a and b are defined with the following boundary conditions :

$$\dot{\alpha}(0) = \dot{\alpha}(T/2) = 0 \tag{E.55}$$

$$\alpha(0) = -\alpha(T/2). \tag{E.56}$$

The first conditions on $\dot{\alpha}$ yields

$$b = \frac{a\xi\omega_0}{2\nu}, \tag{E.57}$$

$$T = \frac{4\pi}{\sqrt{4 - \xi^2}\omega_0}. \tag{E.58}$$

This last condition states that the forcing (E.50) leads a periodic motion with a period equal to $2\pi/\nu$. This period corresponds to the critical one that produces the highest response in α, i.e. a resonance in the amplitude. We remark, that in the limit of small ξ, the swinging policy drives the swimmer to the optimal thrust, as the swinging period tends to the optimal $T^\star$.

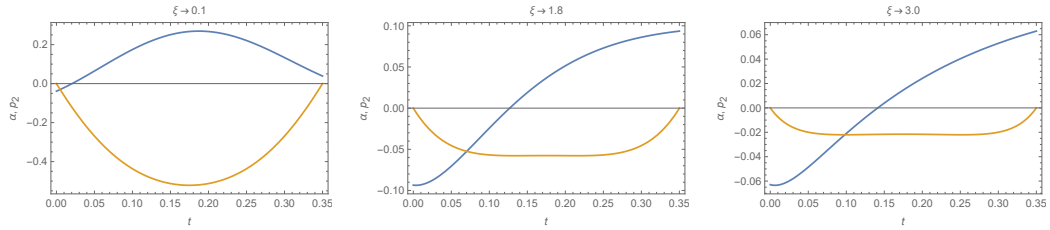

Figure E.2 – temporal evolution of $\alpha(t)$ (blue) and $p_2(t)$ (orange), obtained with $T = 0.7$. $\omega_0 = 12.5\ \mathrm{s}^{-1}$, $\Lambda = 0.1$

.

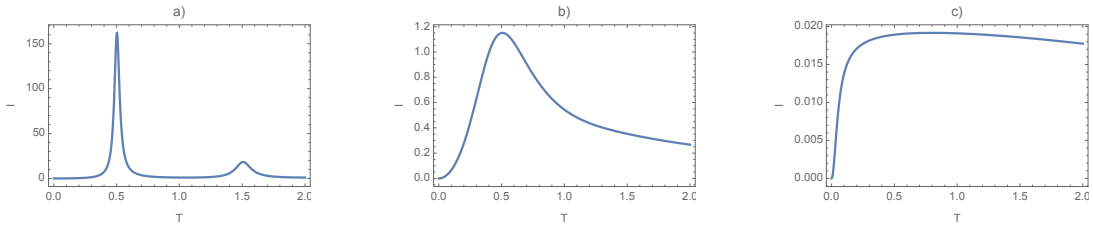

Figure E.3 – Value of the dimensionless thrust as function of the period T, for various values of ξ : a) $\xi = 0.1$. b) $\xi = 1.2$. c) $\xi = 10$

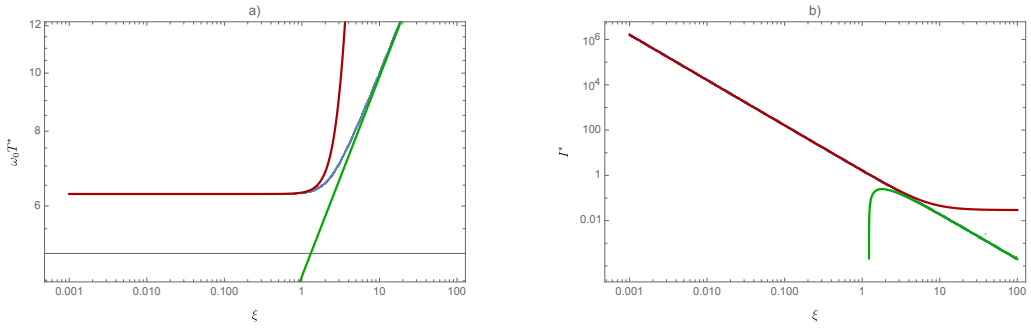

Figure E.4 – a) Optimal dimensionless period $\omega_0 T^\star$ as function of ξ. b) Optimal dimensionless force $I^\star$ as function of ξ. The numerical solutions from Eq. (E.29) are drawn in blue. The small damping limit is the red line and the large damping asymptotics are shown in green.

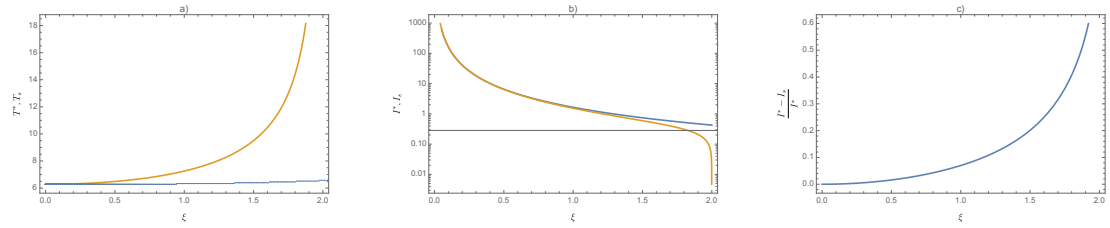

Figure E.5 – a) Comparison between $T^\star$ (blue) and T_s (orange). b) Comparison of the thrust I obtained for $T = T^\star$ (blue) and $T = T_s$ (orange). c) Relative error between the optimal thrust I^* and the resulting thrust obtained by the swinging approach using $\dot{\alpha}(0) = 0$. Here $\omega_0 = 12.5\ \mathrm{s}^{-1}$, $\xi = 1.2$ and $\Lambda = 0.1$.

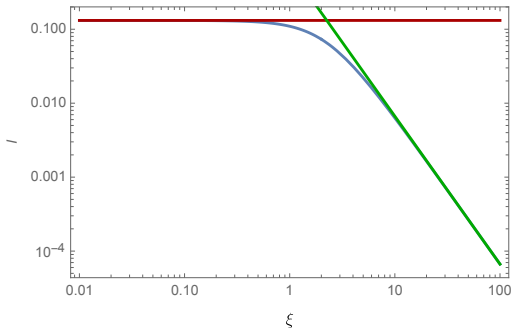

Figure E.6 – Dimensionless thrust I for a slow servomotor as function of the damping parameter ξ (blue). We show the limits of small and large damping in red and green respectively.