UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Análisis numérico de métodos de control activo aeroelástico

Trabajo Fin de Máster

Máster Universitario en Ingeniería Aeronáutica

AUTOR/A: Navarro García, Luis

Tutor/a: Quintero Igeño, Pedro Manuel

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

FINAL YEAR THESIS

Numerical Analysis of Active Aeroelastic Control Methods

Author: **Luis Navarro García**

Tutor: **Pedro Manuel Quintero Igeño**

Master's Degree in Aeronautical Engineering

Higher Technical School of Design Engineering

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

September 2023

A mi familia y amigos, en especial mis padres, por motivarme en la idea de continuar mis estudios de máster y hacerme la persona que soy. A la ciudad de Nápoles, su gente, y mis compañeros de viaje, por acogerme durante la última etapa de mis estudios y hacer de este proceso algo mucho más llevadero convirtiendolo en parte de una gran aventura.

Agradecimientos

Quería agradecer a Andrés Cremades que, no solo desde que el proyecto era solo una idea me regalo su tiempo para ayudar a madurar la propuesta y para encontrar un tutor, sino que además me acompañó en todo el proceso y me otorgo conocimientos vitales para el desarrollo del proyecto. Por supuesto, el agradecimiento a mi tutor Pedro Quintero, por su atención constante y gran dedicación al trabajo, mostrando su apoyo constante, ayudando a centrar ideas, y aportarme todo el material y conocimiento necesario para poder realizar el trabajo. También agradecer al departamento Clean Mobility and Thermofluids *CMT*, por facilitarme las licencias de software vitales para hacer el projecto y los conocimientos sobre los que construir el trabajo. Por último, agradecer a la Universitat Politècnica de València y sus profesionales el buen trato y gran esfuerzo en proveernos a los alumnos de una buena educación, por potenciar un espíritu de innovación y empujarnos a desarrollar nuestras habilidades fuera del aula.

Abstract

A little theoretical background of what Artificial intelligence is, Machine Learning, and the thrilling possibilities that Deep Learning offers give way to an explanation of how Reinforcement Learning works and all its possibilities. With the basic knowledge of aeroelasticity, the idea arises to develop a Reinforcement Learning based algorithm to control the aeroelastic phenomena present in a flat plate. For this, a stack of software will be configured in which multiple technologies will get combined to develop a reduced order aeroelastic model, train a Reinforcement Learning algorithm, and test its capabilities in a high precision StarCCM+ simulation.

Keywords: **Artificial Intelligence, Machine Learning, Deep Learning, Reinforcement Learning, Aeroelasticity, Modelling, FMI, Docker, Flask, REST API, StarCCM+, CFD, Python, RLlib, Neural Network, Proximal Policy Optimization, Control, PID**

Resumen

Un breve antecedente teórico sobre lo que es la Inteligencia Artificial, el Aprendizaje Automático y las emocionantes posibilidades que ofrece el Aprendizaje Profundo, da paso a una explicación de cómo funciona el Aprendizaje por Refuerzo y todas sus posibilidades. Con el conocimiento básico de la aeroelasticidad, surge la idea de desarrollar un algoritmo basado en Aprendizaje por Refuerzo para controlar los fenómenos aeroelásticos presentes en una placa plana. Para ello, se configurará un conjunto de software en el que se combinarán múltiples tecnologías para desarrollar un modelo aeroelástico de orden reducido, entrenar un algoritmo de Aprendizaje por Refuerzo y probar sus capacidades en una simulación de alta precisión con StarCCM+.

Palabras Clave: **Inteligencia Artificial, Aprendizaje Automático, Aprendizaje Profundo, Aprendizaje por Refuerzo, Aeroelasticidad, Modelado, FMI, Docker, Flask, API REST, StarCCM+, CFD, Python, RLlib, Red Neuronal, PPO, Control, PID**

Contents

List of Tables

List of Figures

Nomenclature

Acronyms

2D Two Dimensional

AI Artificial Intelligence

ANN Artificial Neural Networks

AOA Angle of Attack

CFD Computational Fluid Dynamics

DL Deep Learning

DNN Deep Neural Networks

DP Dynamic Programming

FMI Functional Mock-up Interface

FMU Functional Mock-up Unit

GPI Generalized Policy Iteration

i.e. That is

IDE Integrated Development Environment

LSTM Long Short-term Memory

MC Monte Carlo Methods

MDP Markov Decision Process

ML Machine Learning

PDE Partial Differential Equation

PID Proportional, Integral, Derivative controller

PPO Proximal Policy Optimization

RLlib Reinforcement Learning Library

RL Reinforcement Learning

TD Temporal Difference

TRPO Trust Region Policy Optimization

UI User Interface

UPV Universitat Politècnica de València

Chapter 1

Introduction

This chapter focuses in introducing the context in which the project is developed and its main characteristics.

1.1 Motivation

On the one hand, aeroelastic problems have been traditionally dealt with by exhaustive numerical analysis of the phenomena for every given structure of interest and, then, the design process of the structure was heavily constrained by the appearance of these different phenomena. This passive method usually meant a compromise in mass, which in turn affects the efficiency of the overall vehicle. In recent times, there has been a great interest in controlling these phenomena actively, with the idea of improving the performance of a given structure against certain conditions and improving safety. On the other hand, the rise of AI has been exponential, the idea of being able to perform this active control using data driven methods arises as the fields of control, fluid dynamics, and AI have been linked previously in many different contexts, like reduced wake turbulence with active control, and a combination of these technologies could prove beneficial in the future.

1.2 Description

The project focuses in the creation of a workflow that permits the use of AI in a CFD environment that helps evaluate the performance of the different AI models created, in this case for the control of aeroelastic phenomena in a 2D flat plate. The CFD calculations are performed in Siemens' *StarCCM+* [15], and the AI development is carried out in *Python* [16]. Given the high computational costs of CFD calculations, the AI models have to be trained using a reduced order model that allows for a faster calculation time, essential for data driven approaches, which is also created in Python. The interaction between the commercial *StarCCM+*software and Python is carried out using the FMI standard, compatible with *StarCCM+* and accessible in *Python* by the use of the library *PythonFMU* [17]. This interaction is vital for the evaluation of the AI models created. For this project, the main focus is the use of *Reinforcement Learning* as the method to develop the AI algorithms. This method focuses in creating an agent that will interact with an environment, which in this case will be the reduced order simulation, from which it will receive a user-configured reward that will evaluate how good its actions are in order to learn an optimal behavior i.e. a control function.

1.3 Objective of the project

As the project is research focused, the main objective is figuring out the potential capabilities of AI as a technology that can be paired with fluid dynamics and control by performing a test using a 2D flat plate aeroelastic model controlled by an AI based algorithm. For this, several tasks have to be cleared in order to complete the project. First, the coupling of the CFD software and the AI algorithms is key, and it has to be able to send data back and forth in a ordered way. Then, as this project specifically attempts at developing a control algorithm using RL, it's important to generate an environment that contains the physical properties wanted to simulate and that is computationally light enough to use it to train an agent. Finally, the correct selection and training of the RL algorithms is vital to understand the possibilities that this technology can bring for this kind of problem.

1.4 Justification

The possibilities that a technology like this could bring into the field of aerodynamic design, structural design and vehicle performance in terms of efficiency could prove as a great addition in times where, specially aviation, is target of many ecological regulations for the high degree of emissions released. AI for fluid dynamics applications is not just applicable to aeroelastic control, but to turbulence control and modelling, acceleration of CFD computation and many other examples, thus, proving as a really interesting topic to dive in.

1.5 Past work

It has a great influence on this project the article by Torregrosa et al., 2021 [14]. The article focuses on the creation of a nonlinear reduced order aeroelastic model based on a neural network, which is particularly useful in order to use *Reinforcement Learning* as discussed in previous sections. This article serves as the base in which the project starts developing. Another article of great influence to the project is by Hatledal et al., 2020 [17]. The code developed for the paper is key to establish the interaction between the commercial CFD software of choice and Python by means of the FMI standard [18].

Chapter 2

Theoretical Study

In this chapter, the necessary theory to understand RL and its application to control aeroelastic phenomena is discussed. A brief understanding of Computer Fluid Dynamics will be explained and the necessary modelling of the aeroelastic problem to solve will be developed.

2.1 What is AI?

[19] Less than ten years after cracking Enigma, the Nazi encryption machine, Alan Turing asked a simple question. "Can machines think?". In his paper "Computing Machinery and Intelligence" [20], released in 1950 and the Turing Test, a test that determines if a machine can behave in a way that is comparable to human intelligence, Alan Turing outlined the principles and objectives of AI.

Although there isn't yet a precise definition of AI that is widely recognized, one way to think of it is as the area of computer science that attempts to emulate or replicate human intellect and behavior in machines. The fact that the definition of AI provided above does not specifically address what constitutes intelligence in a computer raises a significant issue regarding whether it adequately explains what AI is.

In the book *Artificial Intelligence: A Modern Approach* [21], Stuart Russel and Peter Norvig say that AI is "the study of agents that receive percepts from the environment and perform actions" and they explore four traditional approaches to the definition of AI: thinking humanly and rationally, and acting humanly and rationally.

The first two cover the thought process and reasoning and the next two deal with the behavior.

Another definition comes from Patrick Winston, the former Ford professor of artificial intelligence and computer science at MIT, he defines AI as "algorithms enabled by constraints, exposed by representations that support models targeted at loops that tie thinking, perception and action together"

Both definitions mentioned above may seem pretty abstract to the average person, but they help establish the field as an area of computer science.

2.1.1 How AI is used

AI can be divided into two categories that cover a huge amount of topics: Artificial General Intelligence and Narrow AI.

One may say that Artificial General Intelligence is centered on the AI that appears in science fiction films. This type of artificial intelligence aims to develop a general intellect that, like a human, can be used by a machine to tackle any problem. Many academics seem to view this subject as their ultimate objective, but it has proven to be a very challenging endeavor and it does not appear that it will become a reality any time soon.

Most AI based applications that can be seen nowadays are in the field of Narrow AI and many examples can be found, like *Siri*, or the recommendation algorithms from *Youtube* or *Netflix*. Narrow AI typically mimics human intelligence within a constrained environment. Although they may appear intelligent, they typically function under a number of restrictions and limits that not even the most basic human intelligence compares to. It is typically developed to focus on a specific task exceedingly well.

The most successful application of AI to date is undoubtedly Narrow AI, which has seen tremendous growth over the past ten years, as may be seen from the examples in the previous paragraph.

Modern Narrow AI is mostly powered by advances in Machine Learning, particularly one of its sub-types, deep learning. When considering AI, those two are typically the first names that come to mind, and it can occasionally be challenging to distinguish between the ideas of AI, Machine Learning, and Deep Learning. As stated in the beginning of the text, Deep Learning is essentially a subset of Machine

Learning, which is one of the AI methodologies. [22]

The project was developed utilizing machine learning techniques, therefore comprehension of machine learning and its various forms is essential.

2.2 Machine Learning (ML)

[19] The focus of ML is on developing new algorithms. The simplest and shortest definition of an algorithm is a set of instructions that produces a solution to a problem.

A programmer can develop an algorithm by setting up the rules that define the algorithm's behavior. For instance, if a program is written to output the values of a linear function for the first 100 positive integers, given a specific slope and intercept previously entered by the user in the terminal, it will first acquire the two constants and store them in a variable, then compute the one hundred values, and finally print all of those values in the terminal.

However, in ML, the rules are created by the computer, and the computer is given a number of tools that enable it to learn from data without the need to program each step of the process. As a result, the program described in the previous paragraph can be solved as an example of ML. Assume that after running the method previously described, the first 100 numbers were recorded and tagged as "inputs" and the output of the terminal was saved as "output". In this instance, we are aware of the end goal but are unaware of the process used to produce it. A solution utilizing ML would be the development of a model that receives input and output data and learns how to produce an output that is comparable to that data by adjusting its parameters.

The aforementioned issue is just used as an illustration to show how "traditional" programming and machine learning take quite different approaches. But that issue may be resolved with as little complexity as a regression, and machine learning can go far further. Because object detection in computer vision is a task that humans can perform quite easily but that is challenging to teach a computer how to accomplish using "traditional" programming, current solutions to problems that were previously intractable have been significantly influenced by ML.

A successful ML algorithm can do many different things, in a recent research brief about AI and the future of work [23] it is said that "the function of a machine learning system can be *descriptive*, meaning that the system uses the data to explain

what happened; *predictive*, meaning that the system uses the data to predict what will happen; or *prescriptive*, meaning that the system will use the data to make suggestions about what action to take".

There are three categories for ML:

Supervised ML models get knowledge from labeled data, which enables them to develop and become more accurate over time. The issue mentioned at the beginning of the section, or a model trained with images of dogs and cats that have been tagged by people and afterwards it is able to differentiate between the two on its own, are examples of supervised machine learning.

Unsupervised ML programs look for patterns in unlabeled data. Finding patterns that individuals aren't necessarily looking for or that they aren't really capable of perceiving on their own is made possible by this form of machine learning. The system that recommends products after a user buys or adds another item to their cart on websites like *Amazon* is a typical example of an unsupervised ML software. This software is aware that customers who purchase one product are likely to purchase any of the recommended products as well.

Reinforcement learning works by creating a reward system. RL learns through experimentation what works and what doesn't based on how good the reward received was in order to choose the best strategy for an action. It is frequently used to train models to play games, some of which play even better than people, or to make possible self-driving cars by informing the vehicle when it made the right choices and guiding its long-term learning. The field of RL can be of great interest due to the fact that it can learn complex policies in environments with large dimensions and operate over realistic or highly accurate simulations where data may be hard to get or interpret. A later chapter will go into greater detail on the subject of RL.

The development of computer power is one of the factors that has led to today's excellent ML models and the exponential expansion of the field. The general public now has access to better processors as a result, which has helped the AI community grow significantly and lead to a lot of open-source research and the advancement of the field overall.

As was previously mentioned, ML models learn from examples, which is essentially data, which brings up the other major factor that has contributed to this technology's rapid growth: the vast amount of data that is readily available. Data gathering in a world where the internet permeates every part of daily life means having access to a wealth of knowledge on consumers and many other topics. With ML, this enormous amount of data can be used for research and business.

Google Translate is one example of how such a large amount of data has been used. In order to provide accurate translations, it was trained on the vast volumes of data available on the web and in several languages.

In addition to performing some activities more efficiently than people, or automatically, ML models have also enabled various new tasks, such as *Google Search*, to be carried out.The amount of information that can be browsed in no time and the understanding of the information that might be the most relevant to the user are simply things a human cannot do. In other words, it is not an example of computers putting people aside as is frequently feared, but rather an example of computers doing things that would never have been possible if done by humans.[24][23]

Many different types of algorithms and methodologies are derived from ML, many of which do not fall inside the scope of the project. Natural Language Processing is one of many diverse disciplines of research and applications that seeks to comprehend human spoken and written language in order to produce intelligible text, translations, and even conversations, being great examples *Siri* or *Alexa*.

Uses of ML that are worth mentioning are: recommendation algorithms, image analysis and object detection, fraud detection, chatbots, self-driving cars and even medical diagnostics and imaging.

2.3 Deep Learning (DL)

[19] One particular type of ML that has shown exponential growth in recent years is deep learning. This kind of machine learning is based on how neurons work in the human brain, where each one computes a result from its input and passes that result to the subsequent series of neurons. Perceptrons are another name for these. Given that the implementation of these algorithms uses high level approaches given by various ML programming frameworks, the talk that follows will be focused on ideas that help people comprehend neural networks in a fundamental way.

2.3.1 Neurons and Artificial Neural Networks (ANN)

If only one neuron is considered, the computation inside of it would consist of a linear combination of its inputs, which would then be sent to an activation function before the neuron's output. It's noteworthy to note that a neuron without an activation function is just a linear regression, which would become a multiple linear regression if there were numerous inputs. In the figure 2.1 the basic structure of a neuron can be observed.As can be seen from the figure, a neuron can receive any number of inputs, each of which will be included in a weighted sum to which a bias term will be added. The output of this computation is then passed to the activation function, and the neuron's output is then considered to be the activation layer's output. Later on in the paper, activation operations will be discussed. As was also indicated previously in the chapter, a neuron "learns" by changing its weights and bias in order to fit an input to a desired output.

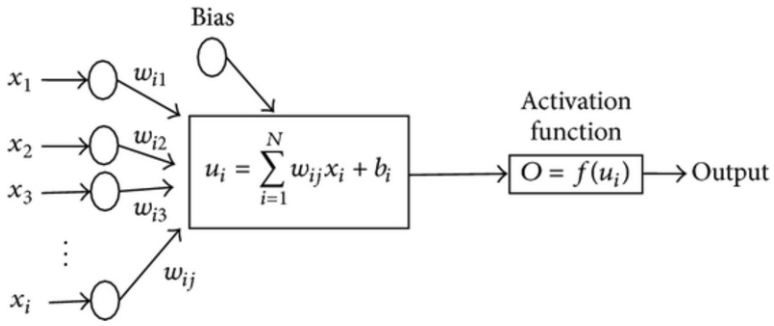

Figure 2.1. The basic structure of an artificial neuron [1].

A rudimentary knowledge of what a neuron is may be gained from this description, therefore judging by its name, an ANN is a collection of neurons that communicate with one another. An input layer, certain hidden layers (which serve as the intermediate levels), and an output layer make up a standard NN. As previously said, this structure was modeled after the human brain, and a simple diagram is shown in the figure 2.2.

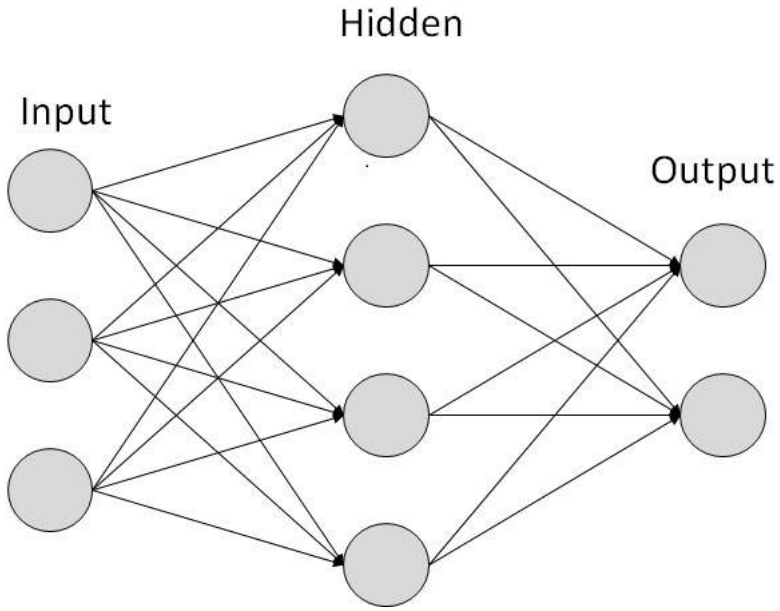

Figure 2.2. Simple ANN [2].

Deep learning comes from the concept of Deep Neural Networks which are NNs that have more hidden layers. An example can be seen in the figure 2.3.

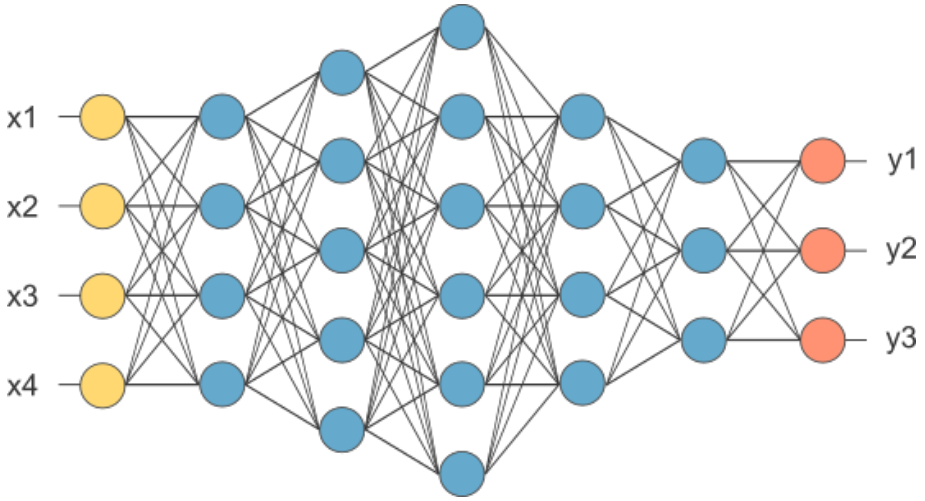

Figure 2.3. Simple DNN [3]

Using DL has the advantage of not really requiring much knowledge of the input because it can immediately learn representations and features from it. Deep learning's widespread use today is also due in large part to the fact that it enables complex models to learn from much larger datasets, which are easy to obtain nowadays. Traditional ML, including NNs, has the issue of overfitting, which is essentially adjusting your model too much to your training data and results in poor performance when receiving inputs that have not previously been seen. This makes generalization often difficult to achieve.

A quick look at activation functions and the training procedure should provide a good understanding of how these algorithms operate now that the structure and fundamental principles of NNs have been clarified.

2.3.2 Activation functions

The purpose of a neuron's activation function is to control how the neuron will combine its output. It is also the function that the NN represents, and some of the most common ones are: the *hyperbolic tangent* activation function, the *logistic* activation function and the *rectified linear* activation function [25].

The hyperbolic tangent activation function, represented in equation (1), is a sigmoid function that makes the output of the neuron vary from -1 to +1. Its output can be seen in the figure 2.4.

$$activation = tanh(combination) \tag{1}$$

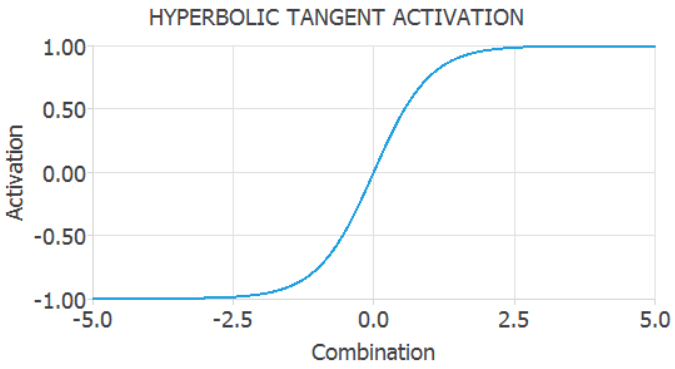

Figure 2.4. Hyperbolic Tangent [4]

The logistic activation function, seen in equation (2), is also a sigmoid function but it varies from 0 to 1. It is represented in the figure 2.5.

$$activation = \frac{1}{1 + e^{-combination}} \tag{2}$$

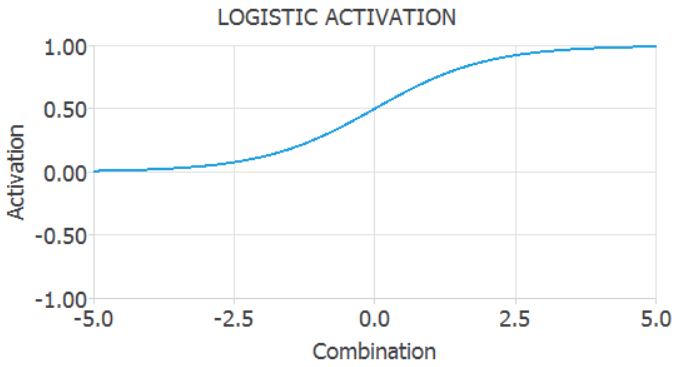

Figure 2.5. Logistic Activation [5]

One of the most, if not the most, common activation functions is the rectified linear activation function, or ReLU. As can be seen in equation (3), it is 0 when the combination is negative and equal to the combination when it is positive. ReLU facilitates the vanishing gradient problem during training, allowing models to be trained more quickly. Vanishing gradient occurs when the gradient rapidly decreases with each iteration to the point where it is difficult to update the parameters in a useful manner; the optimization of the parameters will be covered in the following section. The figure 2.6 shows the output of the function.

$$activation = \begin{cases} 0 & combination < 0 \\ combination & combination \geq 0 \end{cases} \tag{3}$$

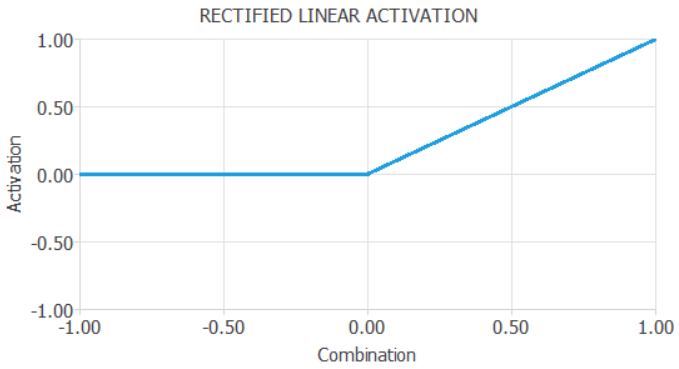

Figure 2.6. Rectified Linear Activation [6]

2.3.3 Training

The use of NNs depends heavily on *loss index*. It provides with a measure of the quality of the representation required to learn and defines the task to be done. The loss index has two terms: the error term and the regularization term, as it can be sen in equation (4) [26].

$$loss\_index = error\_term + regularization\_term \tag{4}$$

The *error* which measures how well the NN fits the data, is the expression's main term. Some of the most important error functions used in the field are: the *mean squared error*, the *normalized squared error*, or the *cross entropy error* amongst many others.

The mean squared error is the average squared error between the outputs of the NN and the real output coming from the data that we want to fit, its expression is seen in the equation (5).

$$mean\_squared\_error = \frac{\Sigma(outputs - targets)^2}{instances\_number} \tag{5}$$

The normalized squared error is obtained by dividing the squared error between the target output and the NN output obtained from data training by a normalization coefficient. The prediction is said to be "on the mean" if the error is near to 1, and it is perfect if the error value is 0. The equation (6) shows the expression for this error.

$$normalized\_squared\_error = \frac{\Sigma(outputs - targets)^2}{normalization\_coefficient} \tag{6}$$

For problems with binary classification, the cross-entropy error is used. This means that the NN should provide an output of either 0 or 1. Using this error has the benefit of penalizing targets that are labeled incorrectly with high error. The equation (7) shows this type of error.

$$cross\_entropy\_error = -\Sigma(target * log(output) + (1 - target) * log(output)) \tag{7}$$

A regular solution is one that for small changes in inputs, small changes in the

output are obtained. For non-regular problems a solution is to control the effective complexity of the NN and this can be done by the usage of a regularization term in the loss index.

Usually the regularization term measures the values of the parameters of a NN. By adding that term to the error the size of the weights and biases are reduced and the response is forced to be smoother. The most used types are L1, equation (8), and L2, equation (9), regularization.

$$l1\_regularization = regularization\_weight * \Sigma\, |parameters| \tag{8}$$

$$l2\_regularization = regularization\_weight * \Sigma\, parameters^2 \tag{9}$$

In order to achieve the desired result, the regularization weight must be adjusted. It must be decreased if the solution is too smooth, and it must be increased if the solution oscillated too much.

The loss index depends on the function that the NN represents. It can be visualized as a surface with the parameters as coordinates as seen in the figure 2.7. The learning process for a neural network consists in tuning the parameters of the NN in order to find the minimum value of the loss index.

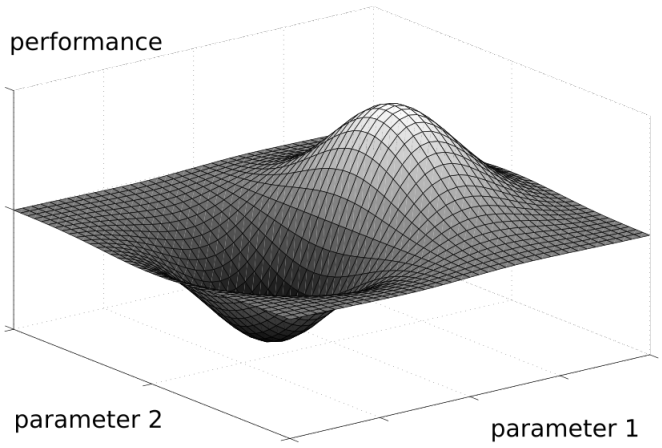

Figure 2.7. Performance Function [7]

As was already established, finding a set of NN parameters that minimizes the loss index constitutes NN training. For this, we know that the gradient will be zero if the NN is at the minimum of the loss index.

Generally speaking, the loss is a nonlinear function of the NN's parameters. It is impossible to find closed optimization techniques for the minima, thus the solution must proceed through the parameter space in a series of steps or epochs. By improving the value of the NN parameters in accordance with a specific algorithm, the loss will reduce at each epoch.

Consequently, in order to train a NN, we begin with a set of parameters that are typically initialized as random integers and create a new set of parameters at each iteration in order to lower the loss index.

Usually the optimization stops when a given criteria is reached, some common are:

- Loss has reached a goal value
- A maximum number of epochs is reached
- The loss improvement in one epoch is less that a set value

The method for changing the NN's parameters is determined by the optimization algorithm. No single optimization strategy can realistically be used for every situation because there are so many different types and they all have different compute and storage requirements. The *gradient descent*, textit *stochastic gradient descent*, or*adaptive linear momentum* algorithm are a few of the most popular examples.

The simplest optimization algorithm is *Gradient descent*, equation (10). Every epoch, this method adjusts the parameters in the direction of the loss index's negative gradient with respect to the parameter it is attempting to optimize. The constant *learning rate* indicates how much the gradient is affecting the new parameters, and it is often set to a certain value prior to optimization or can be changed via line minimization at each epoch.

$$new\_parameters = parameters - loss\_gradient * learning\_rate \tag{10}$$

Stochastic gradient descent, equation (11), updates at every epoch the parameters many times using batches of data.

$$new\_parameters = parameters - batch\_gradient * learning\_rate \tag{11}$$

Since just one training sample is processed by the network at a time, *Stochastic gradient descent* offers various benefits, including making data easier to put into memory. It is quick computationally for the same reason, too. As it changes the parameters more regularly for larger datasets, it can also converge more quickly.

Although gradient descent and the *adaptive linear momentum* optimizer, ADAM, are fairly similar, ADAM employs a more complex algorithm to determine the training direction and speeds up convergence. Additionally, this technique aids in avoiding becoming stuck at the loss's local minima. Since ML frameworks typically already include this functionality, the particular methodology employed by the algorithm is outside the scope of this research.

2.4 Reinforcement Learning (RL)

[19] In the realm of ML, Reinforcement Learning is a tremendously popular topic, and interest in it is only increasing. Its basic definition was provided in a previous section, but the objective of this section is to at least grasp the fundamentals of it.

As a refresher, RL is one sort of ML technique that enables an agent to learn from an interacting environment. In other words, rather of giving the agent access to previously stored data, we instead let it interact with the environment so that it can learn right behavior through trial and error and feedback from its experiences. Both supervised learning and reinforcement learning use mapping between input and output, but in supervised learning the feedback given to the agent is a correct set of actions for a given task, whereas in reinforcement learning, rewards and punishments are used as signals for positive or negative behavior that guide the agent towards learning.

Comparing unsupervised learning and RL, unsupervised learning tries to find similarities in data, however in RL the goal is to find an action model that maximises the total cumulative reward collected by an agent, in the figure 2.8 we see the generic feedback loop for RL [27].

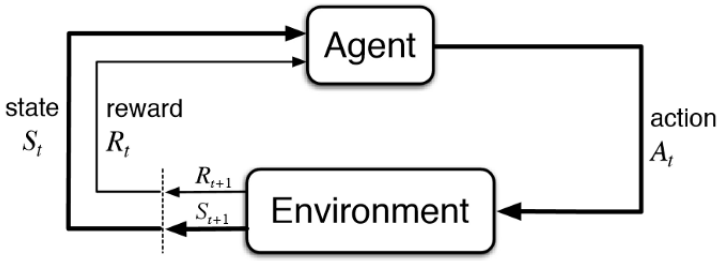

Figure 2.8. Generic Reinforcement Learning Loop [8]

The basic components of RL are:

1. **Environment**: "world" in which agent acts
2. **State**: current data about the agent like, for example, position
3. **Reward**: feedback in the form of points
4. **Policy**: the method that describes what action to take given a certain state
5. **Value**: future reward expected to receive given a certain action in a given state

Games can be a pretty good method to comprehend a RL problem. For instance, you can use a little paddle to manipulate a ball in the game Breakout such that it will bounce into the bricks at the top and award points. If the ball misses, the player receives a penalty. The environment in which the agent acts by moving the paddle to the left, right, or remaining in place in this scenario would be the game screen. By striking the bricks, the agent will be rewarded, and by missing the ball, they will be penalized. The states represent the agent's physical location in the world, and winning the game results in the agent's overall cumulative reward. A screenshot of how the game Breakout looks is given at figure 2.9.

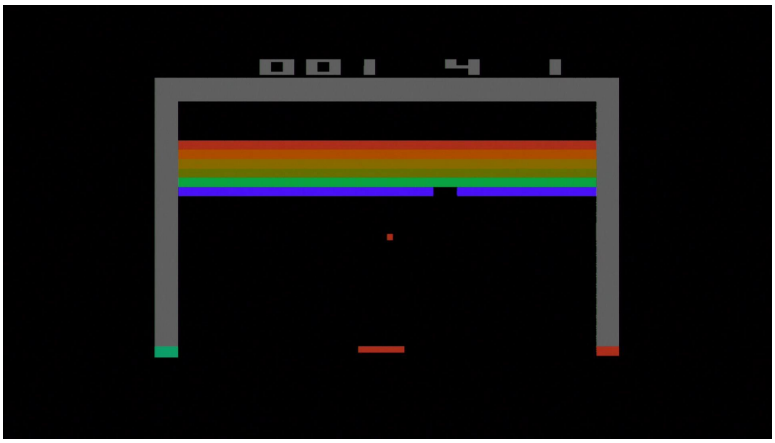

Figure 2.9. Atari Breakout (2600) Screenshot [9]

The agent is faced with the issue of exploring new states while still maximizing the overall reward in order to construct an ideal strategy that behaves as it should. The issue is called *Exploration vs. Exploitation*. In order to strike a balance between the two, the agent may need to make some short-term compromises in order to gather additional data that will help him or her improve the policy and make the best decision possible in the long run. Future parts will touch on this subject and provide further detail.

2.4.1 Markov Decision Process (MDP)

Markov Decision Process is the primary abstraction used in Reinforcement Learning. The MDP is a traditionally formalized sequential decision-making process where actions made affect not just immediate rewards but also following states and, consequently, future rewards. The agent decides an action to make depending on the process's current state at each time step. This action receives a reward and moves the

agent into the next stage. It is crucial to understand that there is always a possibility that a particular action may result in a particular state, known as the state transition probability. [28]

The *Markov Property* is a critical premise. If a state possesses the Markov Property, which means that it does not depend on earlier states and that the current state has the knowledge required to predict the state transition, it is referred to as a Markov state.

With the approach used to solve the issue for this project, it will be clear that the Markov Property is satisfied since, when a simulation run is performed, the changes caused by a specific action can be calculated using the available data.

So, an MDP is defined as a tuple of size 4 (S, A, P_a, R) where S is the finite set of different states, A is the different possible actions, P_a is the state transition matrix and R the reward function.

The agent's objective is to maximize the total reward it obtains from the entire series of acts it does. The equation (12) depicts the sequence of reward obtained over the trajectory. [29].

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{12}$$

G_t is the return.γ Gamma is the discount factor that weighs the importance of future rewards vs immediate rewards. The final time step in the entire series of events is denoted by the symbol T. The return delivers the total discounted future benefit from the current time step, therefore it is an accumulation of rewards for the future and does not consider rewards that have already been received.

This function determines whether the present condition is desirable to remain in or not. The *expected return*, which is the anticipated cumulative reward in the future, governs this. The behavior that determined the action to do is referred to as the *policy*, and rewards are dependent on the action the agent takes. A policy is defined as $pi(a|s)$ and maps the likelihood of carrying out each feasible action given a state. The equation (13) provides a definition for the value function.

$$v_\pi = \mathbb{E}[G_t | S_t = s] \tag{13}$$

$v_\pi(s)$ is the value function of the policy π, G_t is the return and s is the current state. Concurrent with equation (13), formally known as *state-value function* for policy π, the *action-value function* gets defined. The action-value is different from the state-value function in relying in both state and action, seen at equation (14).

$$q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a] \tag{14}$$

In the equation (14), the action-value function, $q_\pi(s, a)$ is the return expected starting from state s, taking the action a and following the policy $\pi(a|s)$.

2.4.2 Bellman Equations

The issue of how to optimize the overall return still has to be addressed. To do this, a policy $\pi(a, s)$ must be established that maximizes the cumulative reward $v_\pi(s)$. The state-value and action-value functions must have a recursive relation in order to discover the best course of action because the preceding definitions of state-value and action-value functions cannot be solved numerically. The recursive relations to be presented use the property of the recursiveness of the return as baseline, the equation equation (15) shows the return expressed as recursive expression with itself.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{15}$$

In the scenario when we have a collection of discretized actions, the value function $v_\pi(s)$ must be utilized to calculate the current value of this state under the policy $\pi(a|s)$, as seen in the equation (16). The sum of all the action-value functions, weighted by the likelihood that a given action will have occurred given the policy, will represent the entire value of that particular state.

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s, a) \tag{16}$$

It is key to know, though, that just as $v_\pi(s)$ is dependent on $q_\pi(s,a)$, $q_\pi(s,a)$ is also dependent on $v_\pi(s)$.The action-value function assigns a value to the action when it is taken, and it also assigns a probability to the outcome of the action. The probability of taking a certain action to end up in a certain state is the transition probability, $P^a_{ss'}$. Knowing this, the equation (17) is now the equation for the state-action value function.

$$q_\pi(s,a) = R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v_\pi(s') \tag{17}$$

The equation (16) and equation (17) can be combined in order to obtain a recursive relation between the initial state and the transition one. If it done for both $v_\pi(s)$, and $q_\pi(s,a)$, we obtain the equation (18) and the equation (19).

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \Big(R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v_\pi(s') \Big) \tag{18}$$

$$q_\pi(s,a) = R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a \in A} \pi(a'|s') q_\pi(s',a') \tag{19}$$

These equations are named the *Bellman Equations* and they express the relationship that exists between the value of the current state and the successor one.

It is now necessary to devise a system for identifying the optimal policy, or the one that maximizes the cumulative benefit over time. The value of both value functions is significantly influenced by the policy, which establishes the agent's behavior. It can be said that a policy π is better or equal than a policy π' when the expected return is greater or equal to the one of π' in all states.

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in S$$

There will always be a policy that satisfies the aforementioned claim and is referred to as an optimal policy. There may be several optimal policies, but they will all have the same value function. The optimal state-value function and the state-action value function can be defined as $v_*(s) = \max \pi v_\pi(s)$ and $q_*(s,a) = \max \pi q_\pi$ and finding these, but especially q_*, help finding the optimal policy.

The Bellman equations mentioned before can be rewritten for the case of an optimal policy resulting in the *Bellman Optimality Equations* that can be seen in the equation (20) and equation (21).

$$v_*(s) = \max_a \Big(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \Big) \tag{20}$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s', a') \tag{21}$$

The absence of $\pi(a|s)$ comes due to the fact that, under an optimal policy, simply taking the action that results in the maximum expected return is an optimal policy. When the selection of the action is done as to always obtain the maximum possible value it is called a *greedy* policy. Once $v_*(s)$ or $q_*(s,a)$ are found, solving the MDP is quite simple; acting greedily with respect to the value functions will be the optimal policy.

Dynamic Programming (DP) algorithms are required to determine the optimal policy. However, DP algorithms are not usually practical as they require knowing the perfect model of the environment as well as its computational expense although all other methods that try to find the optimal policy can be seen as an approximation to DP with less computational effort and without perfect knowledge of the environment's model. The goal of DP is to arrange the search for an optimal policy by utilizing value functions.

The first step in DP is being able to compute the state-value function v_π for a policy $\pi(a|s)$. Determining the state-value function for a given policy is called *policy evaluation*. Converting the equation (18), which is one of the Bellman equations into an iterative update rule gives the equation (22) given that the policy is stationary.

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \Big(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \Big) \tag{22}$$

As with all iterative processes, initial conditions are needed in order to being able to calculate. The condition can depend on the problem but, usually, $v_0(s) = 0$ for all $s \in S$. In general $v_k \to v_\pi$ for $k \to \infty$.

Iteratively solving the state-value function is a technique used in policy evaluation to gauge a policy's effectiveness. Finding better policies is aided by this strategy

as well. If the state-value function v_π is fully determined under a random policy π, for each state s and assessment is made to see if an action a that is $a \neq \pi(s)$ and results in a better behavior can be found. It just happens to be that the action-value function $q_\pi(s, a)$ does exactly that, meaning that if there is an action to be selected that provides with more value and follows policy π, it can be said that selecting this action in this state will always provide more value for the whole trajectory. This is $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, where $\pi'(s)$ is the new policy that will provide with a better behavior, so a new and better policy has been found. This is named *policy improvement*. Having said that, selecting the action that maximizes the value through $q_\pi(s, a)$ can be written as seen in the equation (23). This form of achieving maximization is called greedy optimization policy.

$$\pi'(s) = \arg\max_a q_\pi(s, a) \tag{23}$$

As now a way of both evaluating the policy and improving it have been found, they can get used in alternating manner to evaluate the policy, then improve it, and so on. This is called *policy iteration* and can be seen in the equation (24), where the E stands for evaluation, and I for improvement.

$$\pi_0 \xrightarrow{\text{E}} \pi_{\pi 0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} \pi_{\pi 1} \xrightarrow{\text{E}} \pi_2 \xrightarrow{\text{E}} ... \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_{\pi *} \tag{24}$$

However, since the true value function is typically reached after a number of steps when assessing a particular policy, it is not always required to reach full convergence in each step. Only the ordering and tendency of these values, rather than the exact converged value, are truly important for the majority of applications. In this manner, the computational cost is reduced. A special case for reducing the iterations that it takes policy evaluation is called *value iteration*. Value iteration only does one evaluation iteration and then combined with the policy improvement step it reduces the multiple policy evaluations and subsequent policy improvement to one single update. The equation for value iteration is the equation (25).

$$v_{k+1}(s) = \max_a \sum_{s'} P^a_{s,s'}[R_a(s, s') + \gamma v_k(s')] \tag{25}$$

Value iteration and the Bellman Optimality equation are shown to be related. The updating of the value function under the new policy and the improvement of the

greedy policy are combined in value iteration.

The iterative method described before for the interaction between evaluation and improvements called *generalized policy iteration (GPI)* and it is usually seen in Reinforcement Learning methods. The Bellman optimality equations are correctly approximated when the updates begin to stabilize, and the optimal policies and value function are discovered. DP provides a mechanism for consistently identifying the best policies and value functions, but it uses a lot of computational resources because it necessitates many complete sweeps for larger state representations. They also, and it has to be really highlighted, need the whole model of environment. After mentioning this, the following part will present model-free techniques that focus on observing the environment and attempting to predict the value functions and policy based solely on the feedback it provides. *Reinforcement Learning* methods are those that operate without models and are based on experience.

2.4.3 Model free methods

Since the transition probability of the environment is unknown in model-free RL approaches, learning is only based on interacting with the environment. Value-based methods and policy gradient approaches are the two categories of model-free methods. The dynamic programming techniques covered in the preceding section are extended in value function methods. In policy gradient methods, which are a more recent development, the policy space is parameterized and estimated in a direct manner.

2.4.3.1 Value based methods

This part of the document will cover a more traditional approach that originates from dynamic programming to tackle model free problems. *Monte Carlo methods* will be explained before and later the concepts that were explained about DL in a past section will be combined with value based solutions to cover one of the state-of-the-art reinforcement learning methods, *Deep Q-learning*. Value based methods go way beyond what will be covered, but it serves as a way to understand the way they work.

2.4.3.1.1 Monte Carlo methods

Each problem that wants to be solved using these methods must have a finite length because they rely on the expertise gained over an entire episode. With MC methods, value-functions are learned from the returns on a trajectory in the environment as opposed to DP methods, where value-functions are explicitly generated for all states.

Since there is less information available in MC methods than in DP methods, the rewards are in charge of estimating the value function $V(s)$. This is accomplished by first determining all of the gained reward-state pairs amassed over each episode, and then computing the return $G(s)$ for each state throughout the episode. The value function $V(s)$ is approximated after the returns have been calculated by taking the average of the returns for each state that has been returned in an episode: $V(s) = average(G(s))$. One can choose to merely evaluate the value function the first time a state is visited or to take into consideration all of the visits a state has received for each episode since a single state can be visited several times. Usually the first visit approach is preferred and is called *First-visit MC Prediction*. This method necessitates finishing an entire episode before any calculations can be completed because the return calculation requires knowing the predicted returns of the states before it. The computations are now performed going backward from the episode's end to the beginning.

The value that a particular action provides to the return is more important to control problems than the state-value function, hence calculating the action-value function is of greater relevance. Depending on the number of actions accessible, there are significantly more parameters that need to be calculated for the action value function, which is a challenge to overcome. Another issue is that only one action-value is estimated for each state when the policy is deterministic. This is undoubtedly a component of the exploration problem, which will be discussed further. A policy that ensures that all actions in each state have a non-zero probability must be utilized in order to apply First-visit MC prediction to action-value functions.

The method for evaluating and improving the policy is as seen in the DP methods, GPI, and can been seen in equation (26)

$$\pi_0 \xrightarrow{\mathrm{E}} q_{\pi 0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} q\pi 1 \xrightarrow{\mathrm{E}} \pi_2 \xrightarrow{\mathrm{E}} ... \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} q_{\pi *} \tag{26}$$

The evaluation step is the same as what was previously mentioned, but the itera-

tion step is different because, unlike DP methods, where the method and state-value functions are known, MC methods are model free, therefore they lack this information. However, when utilizing action-value functions, the policy can be easily reduced by choosing the action that will maximize reward for each step without having to wait for complete convergence, allowing for abbreviated iterations. This is comparable to value iteration in DP, which improved the policy after a single evaluation step. In MC, the returns from each episode will be used to evaluate each state's policies and subsequently improve each of them.

In order to comprehend some of the approaches utilized by the RL methods, it is required to have some knowledge of the problem of exploration, which has not yet been addressed. In RL, this problem is key as the way of making an agent behave in an optimal way is usually making it learn action values conditional on subsequent optimal behavior, however to obtain optimal behavior there is a need to explore all of the states, which is sub-optimal behavior. A distinction can be made between two methods of evaluation, *on-policy* and *off-policy*. On-policy approaches assess or enhance the current policy that guides decision-making. Off-policy procedures use data that is collected under a different policy from the one being evaluated or improved. An on-policy method that is improved for better exploration is called ϵ-*greedy*. This policy method behaves greedy with a probability of $1 - \epsilon$, and chooses an action at random with probability ϵ. With this method some actions a forced to be sub-optimal and, thus, improving the exploration.

The two policies that make up off-policy approaches are one that strives to be an optimal policy and the other that ensures exploratory behavior. The policy that is learning to be optimal is the *target policy*, the one that ensure exploration is called *behavior policy* and it is the one used to gather data from the environment. This strategy necessitates extra caution because the data collected is not from the target policy, increasing variation and delaying convergence.

As it has been discussed before, and being coherent, the target policy is called π and the behavior one b. The word *coverage* for off-policy approaches refers to the requirement that both policies guarantee the visitation of the same state-action values. Prior to discussing off-policy MC control, MC evaluation is taken into account. Making adjustments to compensate for the discrepancy during learning is necessary due to the disparity between the target and behavior policy. This is achieved by *importance sampling* which tries to determine the values given by a distribution by using samples of another distribution, in this case the two policies. This technique

is used to weight the returns in accordance with the relative likelihood that each of the two policies' succession of states will occur. This weighting is called importance-sampling ratio, seen in the equation (27).

$$p_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{27}$$

An extension to this approach is *weighted importance sampling*. This method is different as it uses a weighted average instead of the absolute one. This can be seen in the equation (28) to calculate $V(s)$ with the returns scaled by the weighted importance sampling ratio. The sum is for the first-visited stated over all episodes.

$$V(s) = \frac{\Sigma p_{t:T-1} G_t}{\Sigma p_{t:T-1}} \tag{28}$$

This latter approach is the one that is almost exclusively used in practice.

Like most methods there is an incremental update rule for the weighted importance sampling ratio, seen in the equation (29)

$$V_n = \frac{\Sigma_{k=1}^{n-1} W_k G_t}{\Sigma_{k=1}^{n-1}} \tag{29}$$

Each update requires knowing the previously applied weights W_k. This is done keeping the previous weights in C_n. The rule for updating V_{n+1} and C_{n+1} are in the equation (30) and equation (31).

$$V_{n+1} = V_n + \frac{W_n}{C_n}[G_n - V_n] \tag{30}$$

$$C_{n+1} = C_n + W_{n+1} \tag{31}$$

Knowing all this, the pseudo code for the Off-policy Monte Carlo control in given, combining everything that has been mentioned. This is shown in algorithm 1.

Algorithm 1: Off-policy MC control [29].

Initialize, for all $\in S, a \in A(s)$:

$Q(s,a) \leftarrow$arbitrary

$C(s,a) \leftarrow 0$

$\pi(s) \arg\max_a Q(S_t, a)$

while *True* **do**

  $b \leftarrow$any policy with coverage of π

  Generate an episode using b:

    $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T, S_T$

  $G \leftarrow 0$

  $W \leftarrow 1$

  for $t = T-1, T-2, ...,$ *down to* 0 **do**

    $G \leftarrow \gamma G + R_{t+1}$

    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$

    $\pi(S_t) \arg\max_a Q(S_t, a)$

    if $A_t \neq \pi(S_t)$ **then**

      exit For loop

    end

    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

  end

end

2.4.3.1.2 Temporal Difference methods

These techniques are very comparable to DP or MC. In contrast to MC, it also exploits experiences and does not require system dynamics. However, TD learns based on the learnt estimates rather than waiting for an entire episode to finish.

TD's policy evaluation is comparable to MC's. While TD updates the value function at every step, MC waits until the end of the episode to calculate the return. In the equation (32) the prediction step for MC is shown right above the prediction step for TD. α is the step-size, that is constant.

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha[G_t - V(S_t)] \\ V(S_t) &\leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \end{aligned} \tag{32}$$

Updating based on existing estimates is called the *bootstrapping* method. The TD equation above is a *one-step TD* as it only bootstraps one step ahead, which is $V(S_{t+1})$. The term $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is the error between the old value and the estimate of the new value, it is called *TD error* δ and is term that is used many times in RL.

TD on-policy uses the TD technique of evaluation while still adhering to the GPI pattern, just like DP or MC did. Similar to MC techniques, the evaluation is carried out using the action-value function rather than the state-value function. The equation (33) displays this.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{33}$$

The sequence of elements S_t,A_t,R_{t+1},S_{t+1},A_{t+1} that can be seen at equation (33) gives name to this on-policy TD method: *SARSA*.

There is also an off-policy approach for TD, which is one of the most used ones in RL. It is known as *Q-learning*. Q-learning bootstraps like SARSA does, but its off-policy behavior is mostly influenced by the $\max_a$ procedure. The update rule for Q-learning is shown in the equation equation (34).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{34}$$

The behavioral policy in off-policy TD, Q-learning, is often ϵ-greedy for selection the action A_t, the target policy is greedy. Q-values are stores in a tabular manner.

The majority of value-based DL RL techniques are built upon Q-learning. To do this, DL is used as a function approximation.

2.4.3.1.3 Deep Q-Learning

This technique uses DL algorithms to estimate Q-values. However, carrying out this duty is not simple, and unless some changes are made, the results will be poor. As has been mentioned, RL relies on a reward signal that is frequently sparse, noisy, or even delayed. Given that the majority of DL approaches rely on directly obtained feedback, the delay between the action and the reward can be quite problematic for DL. The high correlation in data for RL is another issue, as the state that the environment typically returns depends greatly on the state that came before. Additionally, because DL approaches assume a stable data distribution, the change in policy means that the data distribution will also vary, which is not what is desired.

The update rule for Deep Q-learning is similar to the one of the Q-learning method, equation (34), but the notation changes slightly. This can be seen in equation (35).

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s'a') - Q(s,a)] \tag{35}$$

For being able to apply DL as a function approximator, the action-value function $Q(s,a)$ has to be parameterized: $Q(s,a;\theta) \approx Q(s,a)$. The weights θ are used to parameterize $Q(s,a;\theta)$. Updating the weights is done by the use of a loss function where a gradient descent method or similar can be used. This loss function is defined in the equation (36).

$$L_i(\theta_i) = (y_i - Q(s,a;\theta))^2 \tag{36}$$

The i in L_i and y_i denotes the iteration, where L_i will be the loss and y_i the target for each iteration. In this case, the target is the left side of the TD error, $y_i = r + \gamma \max_{a'} Q(s'a';\theta_{i-1})$. In order to stabilize network updates and lessen the problems with non-stationarity and correlation that were covered at the beginning of the Deep Q-learning section, it must be emphasized that the target is always kept

stable in relation to older parameters. In order to differentiate the loss function with respect to the network weights θ_i so that they can be updated, as explained in the DL section, the equation (37) can be seen.

$$\nabla_{\theta_i} L_i(\theta_i) = (r + \gamma \max_{a'} Q(s'a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a, \theta_i) \tag{37}$$

Even when the suggested method is implemented, using Deep Q-learning is still challenging since it still faces the issues with applying DL to Q-learning that were raised at the beginning of the section. However, this approach has produced a number of cutting-edge outcomes, even outperforming humans in a number of Atari games.

The adoption of a *replay memory*, which saved actions and transition states from the past so that learning could occur by sampling from this database, was crucial to overcoming the Atari difficulties. This technique, known as *experience replay*, allowed for the random selection of sampling from the database, breaking the correlation and thus assisting with non-stationarity. Additionally, a generalization across games was accomplished by employing a ″Convolutional Neural Network CNN as the function approximator to take input from the screen and interpret it. These kinds of solutions are specifically designed to use Q-learning to each type of problem in order to appropriately approach the constraints and the objective that it is seeking to achieve. The programmer must be proficient in these types of procedures in order to grasp their drawbacks and work around them since no generalization can ever be applied.

2.4.3.2 Policy Gradient methods

The discussion up to this point has solely discussed techniques that work by estimating the value function, which means that the policy acts in the direction of greatest return. The strategies to be offered right now directly improve the policy: $\pi(a|s, \theta) = Pr\{A_t = a|S_t = s, \theta_t = \theta\}$. In which θ are the wights of the policy. In order to optimize the policy, equation (38) is given. α is the step size and $J(\theta_t)$ is an arbitrary performance measure. The underlying function approximation must be differentiable to allow for optimization using the gradient.

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{38}$$

The arbitrary performance metric is frequently a value function. A set of techniques known as *actor-critic* methods are created when the use of a value function and a policy estimation method are combined. In these methods, the value function is referred to as the critic, dictating how good or bad the state is to be in, and the actor is the policy, which decides which action to take.

When compared to greedy policies, which are typically seen in value-based techniques, parameterizing the policy offers certain advantages. The ability to approximate the policy space using a stochastic function is one of the benefits. Stochastic function approximation enables the policy to converge to a stochastic optimal policy, depending on the challenge. Another benefit is that it can occasionally be simpler to estimate the policy itself than to estimate the entire value function for the entire problem. In comparison to value-based techniques, a final benefit is that when optimizing the policy, the action probabilities vary smoothly between updates. Value-based techniques choose a new action right away when one action-value surpasses another, which can lead to chaotic behavior if the policy is greedy.

To provide with weight updates the performance measure has to be defined, an example of which can be the value of the starting state s_0, where $v_{\pi\theta}(s_0)$ is the true value function for the policy π_θ. This can be seen in the equation (39).

$$J(\theta) \doteq v_{\pi\theta}(s_0) \tag{39}$$

For various challenges, the performance measure will vary and can be created as shown. This performance metric, specifically its gradient with regard to the parameters, must be impacted by the policy. From the *Policy Gradient Theorem* that establishes a proportional relationship between the gradient of the performance measure and the gradient of the policy, a definition can be given and it is shown in equation (40).

$$\nabla J(\theta) = \mathbb{E}_\pi[G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}] \tag{40}$$

In equation (40), the S_t and A_t are the samples actions and visited states. G_t is the episode return. If equation (38) and equation (40) are combined, it results in a classic policy gradient method called *REINFORCE* and it is shown in equation (41).

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} = \alpha G_t \nabla_\theta \ln \pi(A_t|S_t,\theta) \tag{41}$$

The parameters in equation (41) move in the direction of space that increases or decreases the probability of repeating that action, and it is weighted by the received return. As a result, activities with high returns receive more updating weight. By dividing the action by the real likelihood of selection, the updates are balanced according to the probability of selection. Otherwise, due to their higher chance of being taken, activities that have already been visited frequently would be updated unfairly.

The method *REINFORCE* uses the whole return at each time step, similar to MC, so updates are only done at the end of each episode.

A *baseline* that the action value can be compared to allows the *REINFORCE* method to be extended. This will ultimately end up in the introduction of *actor-critic* methodologies. The action that is significantly better compared to the other ones is what is important when applying action value for updating the parameters. The variance across states may be substantial if absolute values are employed. The updates are normalized by adding the baseline, which lowers the variance at each update step. The equation (42) shows the updates version of *REINFORCE with baseline* in which $b(S_t)$ is a arbitrary baseline value depending on the current state.

$$\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t)) \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \tag{42}$$

Although the baseline might be considered to be a random value, it should have the characteristic that when the actions are in a high value state, the baseline should likewise be in a high value state, and vice versa. For this, the use of an estimates state value $\hat{v}(S_t, w)$ is a good approach. The w are the weights for the state value estimation. The value function can be approximated using the same methodology as MC as the *REINFORCE* technique is an MC method. The actor-critic approaches that will be covered next are closely related to the usage of an approximated state value function as the baseline.

2.4.3.2.1 Actor-Critic

Despite being extremely similar to the actor-critic technique, the *REINFORCE* with baseline is not regarded as one because the value function is employed as a baseline with respect to the return rather than as an estimate. Additionally, an approach like MC that relies on whole episodes converges quite slowly and is unsuitable for online estimation. Therefore, a one-step actor-critic approach is produced by substituting a value function estimate for the return. The value function is updates with bootstrapping, like in TD, so it has to be recalled that $\mathbb{E}(G_t) = V(S_t) = \hat{v}(S_t, w)$. The one step actor critic method is shown in equation (43). δ_t is the TD error that was discussed in past sections.

$$\theta_{t+1} = \theta_t + \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w))\frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \tag{43}$$

This is an online process that in every step can be updated. The term $(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w))$ is the critic in this case, and is the estimation of the value function. The actor is the policy estimation $\frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$. This method is the basis for actor-critic.

The actor and the critic are typically calculated using a NN, hence employing DL and outlining the fundamentals of policy gradient approaches. There are some obstacles to be addressed in order to apply DL to policy gradient approaches, such as the addition of additional stages to provide stability for gradient updates. This and other information on the use of DL to apply to policy gradient will be discussed in the following section.

2.4.3.2.2 Policy optimization

This section cover two of the most popular policy optimization methods: *Trust Region Policy Optimization (TRPO)*, and *ProximalProximal Policy Optimization (PPO)*.

TRPO For non-linear approximation methods with hundreds of parameters, TRPO offers a strategy that helps assure policy improvement. In order to achieve this, TRPO seeks to identify a means of regularizing the step size, which is necessary for complex non-linear policies. In particular, if the policies are non-linear, the outcome would most likely not converge or produce local optima if a constant step size was utilized. Additionally, it would probably never converge if a step size that is too

tiny were to be utilized.

We're going to add a number of new variables to the ones that were previously mentioned. First up is the *advantage function*, which is frequently used in place of the state-value or action-value functions. The advantage function's objective is to normalize the value of the value function, and in order to do this, it deducts the state-value function from the action-values, as shown by the equation (44).

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{44}$$

The advantage function is A_π, then $Q_\pi(s,a)$ is the action value function and $V_\pi(s)$ is the state-value function. It is important to know that the value functions are discounted.

Another term to introduce is the *policy performance* η, which is defined as the discounted expected reward, seen in equation (45).

$$\eta(\pi) = \mathbb{E}_{s_0,a_0,\ldots}\left[\sum_{t=0}^{\infty}\gamma r(s_t)\right] \tag{45}$$

The authors of the original work [30] employed an extremely helpful identity that captures how well one policy performs in relation to another policy's performance and the advantage function. This is shown in the equation (46), where π and $\tilde{\pi}$ are policies, and A_π is the advantage function determined by π.

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0,a_0,\ldots\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)\right] \tag{46}$$

Then, in order to resolve around states rather than steps, which is more in line with RL, this identity is enlarged. The equation (47) is the result of this, where $\rho_{\tilde{\pi}}(s)$ is the discounted visitation frequency.

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s)\sum_a \tilde{\pi}(a|s)A_\pi(s,a) \tag{47}$$

What the equation (47) means is that when the expected advantage at every state s is not negative, the policy performance of $\tilde{\pi}$ will be at least equal or higher than the policy performance of π.

It should be emphasized that the new policy on the equation (47) is dependent on how frequently the state visits. This is challenging to determine when utilizing data from the previous policy, so an approximation of the frequency of state visits is required. For this, the state visitation frequency that is used is from the old policy $\rho(\pi)$ instead of $\rho(\tilde{\pi})$ to work as an approximation. There is evidence that the derived identity will remain valid for a modest enough policy improvement step, but this necessitates estimating how big the improvement step should be. Using the aforementioned limitations, the equation (48) provides the final local approximate identity.

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a) \tag{48}$$

In the previous paragraph, it was said that it was necessary to understand how to choose a step size that would guarantee that the policy would improve while remaining modest enough to prevent it from being broken. The authors of the paper *Trust Region Policy Optimization* [30] The authors utilize KL divergence between the old and new policy, obtaining the equation (49). KL divergence, which was used to make the theorem useful for the majority of policies, measures the difference between two probability distributions.

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - C\dot{D}_{KL}^{max}(\pi, \tilde{\pi}) \tag{49}$$

$D_{KL}^{max}(\pi, \tilde{\pi})$ is the KL divergence between the two policies, $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$ and $\epsilon = \max_{s,a} |A_\pi(s,a)|$. The lower bound approximation of the performance function is defined using the KL divergence and the variable C combined. Another term needs to be introduced before the application of this equation is covered. If $M_i(\pi) = L_{\pi_i}(\pi) - C\dot{D}_{KL}^{max}(\pi, \tilde{\pi})$, combining it with equation (49), results inequation (50).

$$\eta(\pi_{i+1}) - \eta(\pi_i) \geq M_i(\pi_{i+1}) - M(\pi_i) \tag{50}$$

The equation (50) shows that the actual policy performance will always be higher or equal to that of M_i. So, if M_i is maximized, this will result in a policy performance improvement. This algorithm is called *minorization-maximization (MM).*

As parameterized policies are learned, the previous notations concerning π have to be substituted by the policy parameters θ. The objective function, then, will result

in the maximization of $M_i(\theta)$, parameterized by the policy parameters θ and is shown in equation (51).

$$\underset{\theta}{\text{maximize}} \left[L_{\theta_{old}}(\theta) - C\dot{D}^{max}_{KL}(\theta_{old}, \theta) \right] \tag{51}$$

The authors also pointed out that the updates would be too little to allow for convergence if C were used to penalize the function. Due to the dependence on the KL divergence, finding a constant C that would provide good performance turned out to be somewhat challenging. Therefore, a limit is placed on the Kl divergence rather than penalizing it: $D^{avg}_{KL}(\theta_{old}, \theta) \leq \delta$. The use of average KL divergence must be stressed because maximum KL divergence is unstable. Additionally, the KL divergence has no upper bound, raising the possibility of instability. The Trust Region Policy Optimization's ultimate limitations are a result of this, shown on equation (52).

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\, L_{\theta_{old}}(\theta) \\ &\text{subject to}\, \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta \end{aligned} \tag{52}$$

Now that the theory has been clarified, a useful objective function may be established in order to optimize it using the theorem given by equation (52). From equation (48), if the objective function is rewritten in a more convenient way, the result is shown in equation (53).

$$\begin{aligned} \Delta\eta &= \sum_s \rho_{\pi_{\theta_{old}}}(s) \sum_a \pi_\theta(a|s) A_{\pi_{\theta_{old}}}(s,a) \\ &= \sum_s \rho_{\pi_{\theta_{old}}}(s) \sum_a \pi_{\theta_{old}}(a|s) \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_{\theta_{old}}}(s,a) \right] \end{aligned} \tag{53}$$

The first part, $\sum_s \rho_{\pi_{\theta_{old}}}(s) \sum_a \pi_{\theta_{old}}(a|s)$, gets determined by the episode returns of the older policy. The part to be optimized is inside the brackets, $\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_{\theta_{old}}}(s,a) \right]$. If equation (52) and equation (53) are combined and converting it to a sampled based approach like usually in RL, the equation (54) is obtained.

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\, \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_{\theta_{old}}}(s,a) \right] \\ &\text{subject to}\, \hat{\mathbb{E}}_t \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta \end{aligned} \tag{54}$$

TRPO uses MC roll outs to calculate state visitations, Q-values, and KL divergence estimates in order to maximize this objective function. A conjugate gradient approach is used to optimize the parameters, necessitating a local approximation of both the objective function and the constraint of the KL divergence. TRPO has demonstrated excellent results for continuous state and action problems, including the Atari games stated in the Deep Q-learning section.

TRPO has extremely high computational requirements and is incompatible with various DL techniques. But the Trust Region strategy has succeeded to the extent that the concept of Trust Regions is widely applied in RL settings.

This is where **PPO** is introduced. PPO is less computational expensive method than TRPO is, but is also easy to implement and is built on the foundations of TRPO.

Comparatively to TRPO, which requires more involved techniques like conjugate gradients, PPO is a simplification of the TRPO method that enables the use of ordinary gradient descent methods. PPO is based on two main principles. The KL divergence between two policies imposes a constraint on the update step according to TRPO. PPO, on the other hand, limits the update step based on the probabilities between the existing and new policies, as opposed to applying a constraint. From equation (54), and with the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, a new function is made that instead of using the constraint, limits the update step due to the constraints invoked on $r_t(\theta)$. This can be seen in equation (55)

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, clip(r_t(\theta)), 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right] \tag{55}$$

The adjusted loss function's impact is depicted in the figure 2.10. When a particular action, under a new policy, has a high probability of being carried out in comparison to the old policy and the associated advantage function is positive, when it is not unbounded, it might lead to a significant update step. The side of the step is constrained by the clip bounds in this update rule, which are $[1-\epsilon, 1+\epsilon]$. In this manner, more cautious measures are used. The same situation occurs when a probability ratio is significantly below 1 and there is a negative advantage function.

PPO demonstrated that there were appreciable performance improvements as compared to other policy gradient algorithms in the publication in which it was presented [31]. PPO is a cutting-edge method for policy optimization that's popular in RL when we consider all of the factors mentioned above, including how simple it

is to construct and the flexibility to leverage more traditional gradient optimization techniques.

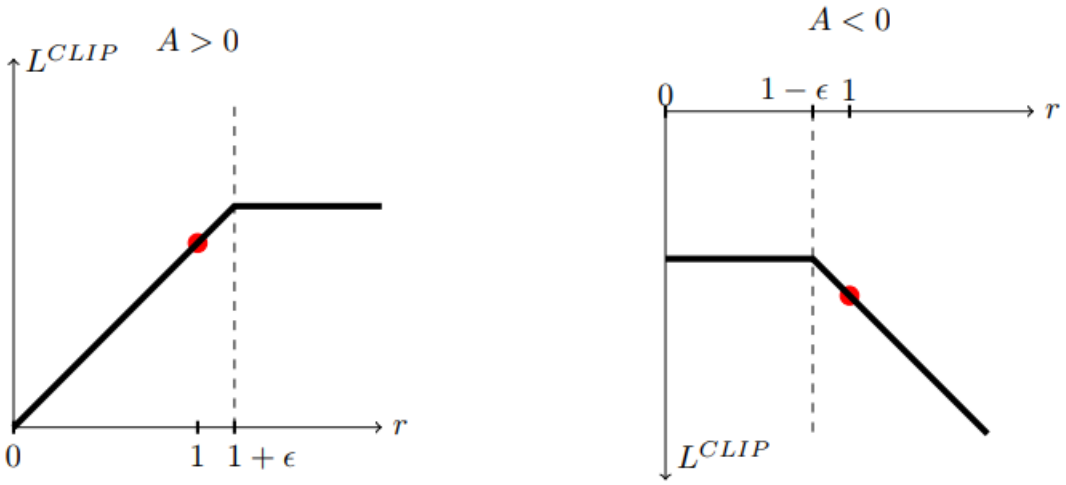

Figure 2.10. Clip parameter illustration [10]

2.4.4 Problems of RL

RL is unstable and struggles to converge to the best outcome. Multi-agent situations, where there are multiple agents rather than just one, make the issue worse. There are a few challenges that must be considered when RL is implemented, and they will be covered next.

The fact that the state and action space expand significantly as the problem complexity rises is what causes the *Curse of dimensionality*. These grow considerably more when the state and action spaces are continuous. It is feasible to generalize using function approximation line NNs, but it is simple to envision that solving large problems fast requires a lot of computational power in addition to better and more efficient function approximation. Usually, this results in higher volatility or other stability problems.

The *explore or exploit* dilemma is a problem that is directly related to the exponential growth of available space. Previously, this issue has already been raised. It takes a lot of states to create a good representation of the environment, but as the algorithm improves and a better and better policy is found, there is a significant possibility that many of the states remain undiscovered and that possibly better policies have not been discovered. In other words, the solution was trapped in a

local optimum. These are, of course, related to the number of states, since the more, the worse. This is where the problem arises since it might be difficult to determine whether it is best to investigate or utilize the available knowledge in the search for the ideal answer.

credit assignment is a further issue. This issue arises from the difficulty in determining which action led to a given reward. This is something that can be partially resolved by reducing the reward's sparsity, but doing so would require overly customizing the reward and providing the agent with too much information, which could prevent the agent from coming up with more creative solutions than it otherwise might. As a result, the use of RL is diminished because this is essentially one of its key attributes.

Also regarding the amount of information is the observability of the system. It is not realistic the the agent has full observability of the system and, thus, most likely the system will be only partially observable. Another of the reasons why this is the case is that having full observability would make the state space huge. And the problem with this was discussed earlier in the section.

2.5 Computational Fluid Dynamics CFD

"Computational Fluid Dynamics (CFD) is the process of mathematically predicting physical fluid flow by solving the governing equations using computational power." [32] CFD is the preferred technique for aerodynamic calculations given the computing power that it's available as of today. The most common CFD tools are based on the Navier-Stokes equations.

2.5.1 Governing equations of CFD

The basic equations are the three laws of conservation:

1. Conservation of Mass: Continuity Equation
2. Conservation of Momentum: Newton's Second Law
3. Conservation of Energy: First Law of Thermodynamics or Energy Equation

According to these principles, in a closed system, mass, momentum, and energy are stable constants. In essence, all resources must be preserved.

Certain physical characteristics are necessary for the analysis of fluid flow with heat variations. The three unknowns from these three fundamental conservation equations that must be simultaneously determined are the velocity $\vec{v}$, pressure p, and temperature T. The two necessary independent thermodynamic variables, however, are p and T.

The conservation equations' final form additionally includes four more thermodynamic variables, including density ρ, enthalpy h, viscosity μ and the thermal conductivity k, final two of which are also transport properties.

Analysis of fluid flow is necessary to determine $\vec{v}$, p, and T at each stage of the flow regime. A key problem is the technique for observing fluid flow that is based on kinematic characteristics. Either Lagrangian or Eulerian techniques can be used to analyze fluid motion.

- The notion behind the Lagrangian description of fluid motion is to follow a fluid particle that is large enough to notice characteristics. It is necessary to compare a particle's initial coordinates at time t_0 to its coordinates at time t_1. It would be nearly hard to follow millions of individual particles through the journey.
- Instead than tracking a singular particle across the path, the Eulerian approach looks at the velocity field as a function of space and time.

2.5.1.1 Continuity equation

The equation of conservation of mass is shown in the equation (56). Where ρ is the density, $\vec{v}$ is the velocity and ∇ the gradient operator.

$$\frac{D_\rho}{D_t} + \rho(\nabla \cdot \vec{v}) = 0 \tag{56}$$

2.5.1.2 Navier-Stokes equation

The conservation of momentum equation is given by the equation (57). Where p is static pressure, $\overline{\overline{\tau}}$ is the viscous stress tensor and $\rho\vec{g}$ is the gravitational force per unit of volume.

$$\overbrace{\frac{\partial}{\partial t}(\rho\vec{v})}^{I} + \overbrace{\nabla \cdot (\rho\vec{v}\vec{v})}^{II} = \overbrace{-\nabla p}^{III} + \overbrace{\nabla \cdot (\overline{\overline{\tau}})}^{IV} + \overbrace{\rho\vec{g}}^{V} \tag{57}$$

The different braces mark:

I Local change with time

II Momentum convection

III Surface force

IV Diffusion term

V Mass force

According to Stoke's Hypothesis the stress tensor $\overline{\overline{\tau}}$ cac be specified as in equation (58)

$$\tau_{ij} = \mu \frac{\partial_{v_i}}{\partial_{x_j}} + \frac{\partial_{v_j}}{\partial_{x_i}} - \frac{2}{3}(\nabla \cdot \vec{v})\delta_{ij} \tag{58}$$

2.5.1.3 Newton's first law of Thermodynamics

The conservation of energy is the first law of thermodynamics. The sum of work and heat added to a system will results in the increment of the energy of the system. As expressed in equation (59), where dQ is the heat added, dW is the work done on the system and dE_t is the increment in total energy.

$$dE_t = dQ + dW \tag{59}$$

A common type of energy equation can be see in the equation (60).

$$\rho \left[\overbrace{\frac{\partial h}{\partial t}}^{I} + \overbrace{\nabla \cdot (h\vec{v})}^{II} \right] = \overbrace{-\frac{\partial p}{\partial t}}^{III} + \overbrace{\nabla \cdot (k \nabla T)}^{IV} + \overbrace{\phi}^{V} \tag{60}$$

The sections marked with braces mean:

I Local change with time

II Convective term

III Pressure work

IV Heat flux

V Source term

2.5.2 Discretization

The numerical solution is a method based on discretization that obtain approximate solutions to complex problems that cannot be solved analytically. The accuracy of the numerical solution highly depends on the quality of the discretization. Many discretization methods exist like finite difference, finite volume, finite element, spectral (element) methods and boundary element.

2.5.3 Mesh convergence

To conduct an analysis the domain of the solution is split into many sub-domains, called cells. The combination of all the cells that form a domain in the computational structure is called *mesh*. In a *mesh* the cells are small enough to apply the mathematical models assuming linearity. This means that areas where the behavior of the fluid is highly non-linear, the mesh will need to have smaller cells

2.5.4 Mesh errors and mesh independence studies

Errors that come from a badly configured mesh are a common issue and result in inaccurate solutions or complete failure of CFD solutions. A pretty common error is having a mesh where the cells are too big and the physics of the flow cannot be captured properly. This is the reason why mesh independence studies are carried out to ensure the mesh is not having a big impact in the final solution. A typical schema for a mesh independence stufy can be:

1. Create a first mesh that visually looks to have enough cells and a dense enough mesh while accurately capturing the geometry.

2. In areas of interest, regenerate the mesh with more cells and a denser mesh. Rerun the CFD analysis and compare the outcomes as necessary. For instance, comparing pressure drops at key locations would be a good way to gauge mesh sensitivity when examining internal flow through a channel.

3. Continue mesh refinement until the outcomes and important physical characteristics (such as pressure drop, maximum velocity, etc.) adequately coincide with the prior mesh and CFD study.

By doing this, mesh structure-related mistakes can be minimized, and the ideal number of components can potentially be reached to facilitate computing.

2.5.5 Convergence

CFD relies on gradually changing results which eventually land on a final solution. An initial guess is employed as the beginning point of a CFD study. The solution field transforms from an initial guess into a final, persistent flow field through numerical iteration. Remember that even while numerical iterations are what lead to the ultimate answer, the mesh still plays a crucial role in the precision of the outcome.

A crucial problem for computational analysis is convergence. The convergence of a non-linear mathematical model of fluid motion is greatly influenced by complex models like turbulence, phase change, and mass transfer. In addition to the analytical solution, the numerical solution also follows an iterative process where solutions are acquired by minimizing errors made in earlier steps. The inaccuracy is identified by the variations between the final two solutions. The result becomes more trustworthy and moves closer to a stable solution as the absolute error is descending.

Until the solution field stops changing, the solving process should be continued. Both steady-state and transient simulations demonstrate this. Convergence must be attained for transient simulations at each time step as if it were a steady-state simulation.

With each iteration, equation residuals change. Convergence is attained when the number of repetitions reaches a threshold. Here are a couple more crucial convergence facts:

- Convergence can be sped up by factors including the Courant number, under-relaxation, and initial conditions.
- A weak mathematical model and mesh can produce inaccurate but converged results; the solution does not always have to be accurate.
- Convergence can be stabilized using a variety of techniques, including first- to second-order discretization schemes, mesh refinement, and adequate mesh quality.
- To avoid ambiguity, make sure the answer may be repeated if necessary.

2.6 Aeroelasticity

Aeroelasticity is a field that focuses on studying problems related to the deformations of structures in an airflow. These deformations interact with the airflow, for example changing the angle of attack of a airfoil and, thus, resulting in a change in aerodynamic forces [11] as seen in the figure 2.11.

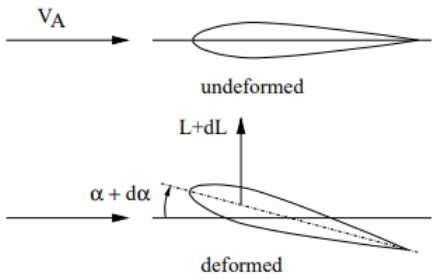

Figure 2.11. Elastic deformation of airfoil [11]

The diagram in the figure 2.12 explains the interaction between the structure and the aerodynamics forces. It can be seen that the diagram shows a closed loop system, thus meaning that there is a mutual dependency between the internal structural forces and the aerodynamics forces.

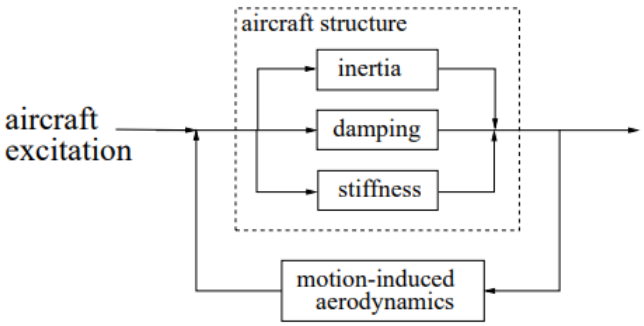

Figure 2.12. Functional diagram of oscillating aircraft structure [11]

For the correct realization of the project it will be necessary to construct an aeroelastic model of a 2D flat plate. The aeroelastic phenomena and the analysis itself will be avoided, as the scope of this project is to develop RL algorithms by the interaction with the aeroelastic model and, thus, the interest remains only in the modelling phase.

2.6.1 Aeroelastic modelling

Considering a generic airfoil like the one shown in the figure 2.13, a general aeroelastic model without damping can be written for an airfoil with chord c. Where w is the vertical displacement. α is the rotational displacement, k_α is the torsional rigidity, and k_w is the vertical rigidity. Thus, having 2 degrees of freedom.

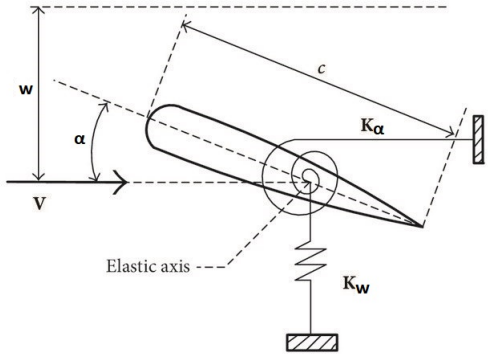

Figure 2.13. 2D undamped aeroelastic model of airfoil [12]

The first step in modelling will be formulating the equations of motion, which can be derived using Langrange's equations shown in equation (2.6.1), being T the total kinetic energy, U the total potential energy and L the generalized forces associated to each degree of freedom.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i$$
$$L = T - U$$

The total kinetic will result in the equation (61).

$$T = \frac{1}{2}m\dot{w}^2 + \frac{1}{2}I\dot{\alpha}^2 \tag{61}$$

The total potential energy is shown in the equation (62).

$$T = \frac{1}{2}k_w w^2 + \frac{1}{2}k_\alpha \alpha^2 \tag{62}$$

For the generalized forces in each degree of freedom Q_w for the vertical one, and Q_α for the torsional one are shown in the equation (2.6.1). Being L the lift generated by the airfoil, and M the moment, both aerodynamic forces.

$$Q_w = L = L(\alpha, w, \dot{\alpha}, \dot{w}, \ddot{\alpha}, \ddot{w}, ...)$$
$$Q_\theta = M = M(\alpha, w, \dot{\alpha}, \dot{w}, \ddot{\alpha}, \ddot{w}, ...)$$

2.6.1.1 Aerodynamics

The aerodynamic model for our problem will be data-based, as explained in Torregrosa et al., 2021 [14]. For that, first a dataset must be constructed using a CFD software like StarCCM+. It is interesting to divide the coefficients in a stationary and dynamic part, both summed giving the total coefficient.

With the data obtained from the CFD simulation, interpolation curves are built that are able to obtain the stationary part $c_{l,m}^{st}$ of the coefficients given a certain AOA α. For the dynamic part, a NN is trained in which the inputs are the average AOA $\overline{\alpha}$, the change in AOA $\Delta\alpha$, the derivative of the AOA $\dot{\alpha}$ and the second derivative of the AOA $\ddot{\alpha}$, and it outputs the dynamic part $c_{l,m}^{dyn}$. The sum of the stationary and the dynamic parts result in the total coefficients use to calculate lift L and moment M as shown in the equation (2.6.1.1)

$$L = \frac{1}{2}\rho_\infty c V_\infty^2 \left(c_l^{st}(\alpha, w) + c_l^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha})\right)$$
$$M = \frac{1}{2}\rho_\infty c^2 V_\infty^2 \left(c_m^{st}(\alpha, w) + c_m^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha})\right)$$

2.6.1.2 Final model

Knowing the total kinetic energy T, the total potential energy U and the generalized forces Q, we obtain the equations for the two degrees of freedom θ and w in the equation (2.6.1.2) and equation (2.6.1.2) respectively and, thus, the aeroelastic model we need for the project.

$$I\ddot{\theta} + k_\theta \theta = \frac{1}{2}\rho_\infty c^2 V_\infty^2 \left(c_m^{st}(\alpha, w) + c_m^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha}) \right) \longrightarrow$$
$$I^*\ddot{\theta}^* + k^*\theta^* = c_m^{st}(\alpha, w) + c_m^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha})$$

$$m\ddot{w} + k_w w = \frac{1}{2}\rho_\infty c^2 V_\infty^2 \left(c_l^{st}(\alpha, w) + c_l^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha}) \right) \longrightarrow$$
$$m^*\ddot{w}^* + k_w^* w^* = c_l^{st}(\alpha, w) + c_l^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha})$$

Chapter 3

Methodology

This chapter shows the process to construct the necessary software and simulations. To understand the though process behind the development of the software and simulations and have a clear idea of how the pieces of the puzzle are put together, two flow diagrams are constructed that put into context the usage of the software, which is more extensively explained further in chapter. The first diagram in the figure 3.1 explains the training architecture of the software. The code is available at the Github repository from reference [33].

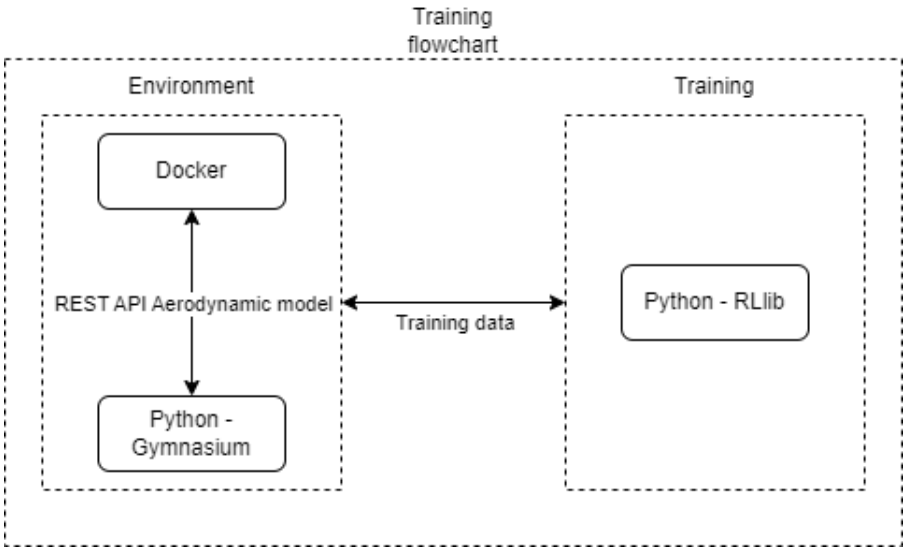

Figure 3.1. Flowchart describing the data flow and software architecture used for training of the RL algorithm

The diagram in the figure 3.2, shows the architecture used for inference and testing, coupling the RL with a CFD simulation in StarCCM+.

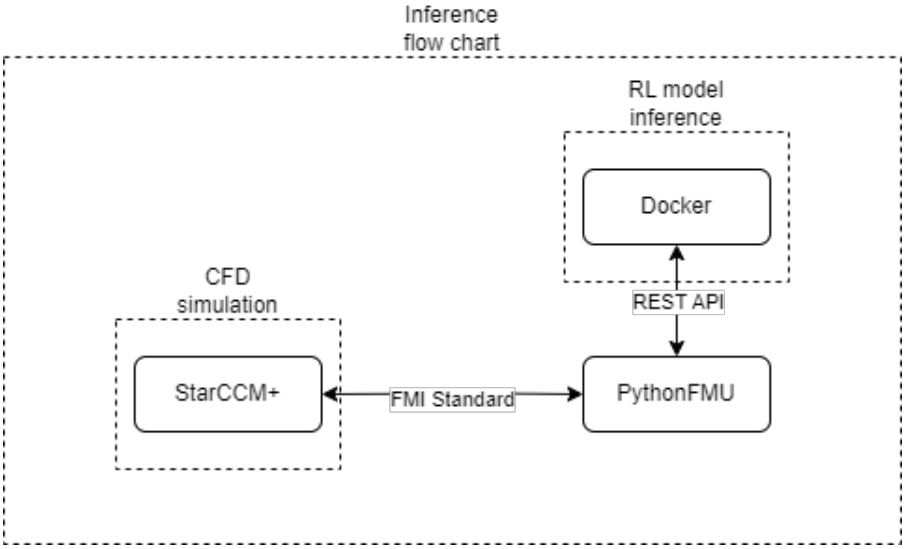

Figure 3.2. Flowchart describing the data flow and software architecture used for inference and testing of the RL algorithm

3.1 Software selection

3.1.1 *RLLib* framework

RLlib is part Ray, which is a framework that "provides a simple, universal API for building distributed applications" [34]. Ray offers simple primitives for creating and executing distributed applications, allowing users to parallelize single machine code with few or no code changes. On top of the core Ray, Ray has a wide ecosystem of tools, applications, and libraries, one of which is RLlib, which enable complicated applications. The framework has a Python, Java, and experimental C# API. The Python API will be used in this project.

No additional description of Ray is required because the majority, if not all, of the code needed to execute the tests is done with RLlib alone. However, excellent documentation that is regularly updated and available online from one of the sources exists. [34].

"RLlib is an open-source library for RL that offers both high scalability and a unified API for a variety of applications" [34]. This library is quite well-liked, and there are instances of it being utilized in organizations like JPMorgan, which uses it to power trading models. [35].

TensorFlow and PyTorch are just two examples of the machine learning frameworks that RLlib supports, however the internals are mostly independent of any of these frameworks.

Thanks to excellent abstraction, the RLlib API is fairly clear and simple, especially for simple scenarios. The library can also be used in a far more low-level manner, offering a platform for experimentation and development.

3.1.1.1 Environments

RLlib is compatible with Gymasium, usually referred as *Gym* which is "a toolkit for developing and comparing RL algorithms" [36]. Gym provides a straightforward API to create your own unique environments or test some already-existing ones. This is done by inheriting a Python class with a series of functions, that are expected to be recognized and accessed later by the training API, and that will have to be overwritten to return a given output that is also expected by the training API and that will contain information about the environment like the reward necessary to find a solution to the problem proposed. This approach offers a straightforward mechanism for abstract RL algorithm implementations to execute the environments. The main functions of this class are the *init* function, the *reset* function, and the *step* function.

The *init* function, as its name says, is run when a new instance of the environment is created, and just for one time. Constants are typically declared in this place, and the state space and action space must both be defined. This function is not expected to return anything.

The *reset* function serves as a way to restart an episode of the environment. All of your variables must be returned to their starting states in this function, and it must also return a set of initial observations that precisely match the state space definition from the initialization function.

The *step* function is the most complex one.This process is in charge of moving the environment's time forward by one step. That implies that this function must calculate the following succession of states. To calculate the rewards corresponding to the new states, the logic for rewards must also be coded inside of this function. The *step* has to return both the new states and the reward, but also two flags called *terminated* and *truncated* that are booleans that indicate when an episode has finalized. *Terminated* means the the agent has reached a terminal state, which could be good or bad,

and *truncated* means that the episode must end due to another constraint like a time limit, or other factors like and unacceptable state that when being reached means the simulation has to finish. Also, like for the *reset* function, the states return have to match the expected format declared in the state space at the the *init* function. The training API will use the step function to give commands to the agents, therefore it must be taken into consideration that the step function also expects the action as an input.

The employment of helper functions is entirely permissible because the environment is specified as a Python class. This means that in order to return what is required from the environment, calls to the helper functions must occur inside the previously described functions, even though tasks are not strictly required to be written inside of them.

The amazing thing about this method is that anything that can be written inside those functions can be a valid environment. As a result, it can be used to play games and make calls to various external apps, making RL solutions for a vast variety of issues available.

3.1.1.2 Algorithms

From extremely basic "hello world" applications to cutting-edge algorithms, RLlib supports many different types of algorithms that are appropriate for many different purposes.

Naturally, several of the various algorithms covered in earlier parts can be found among the numerous others, ranging from policy gradient techniques to value-based models. The well-known PPO algorithm is just one example.

There are countless options for customizing an algorithm. The various hyper parameters that are available for various algorithms can be adjusted using RLlib. For PPO, for instance, we can describe the model's characteristics. NNs are utilized to approximate the policy function or the value function in these models, which are often DL-based. As was already noted, these models may be completely customized, and the library is fully integrated with ML frameworks like TensorFlow, or PyTorch that support this.

Specific algorithms are used by instantiating them with a given configuration and by providing the desired environment to train with. For the project, and a first

approach to solving this kind of problem, it has been opted to use the algorithm PPO, as it has really good convergence for a wide range of scenarios and it is fairly simple to apply.

3.1.1.3 Training

RLlib offers a fantastic training API that is straightforward and simple to use while still allowing access to additional low-level functions.

At a high level, RLlib provides a trainer method for the algorithm class, that is in charge of maintaining the environment interaction policy. The policy can be trained, saved and further methods allow for an action to be computed all inside the algorithm class.

The algorithm class operation will be briefly explained so that you can grasp it a little bit better. As usual, there is a lot more detailed and extensive material available, but understanding how the algorithm class works in depth as a user is not necessary.

RLlib anticipates a number of configuration parameters and the environment we want to train in order to begin utilizing the algorithm. The list of setup parameters is quite long and may seem daunting at first, but there are now well-known parameters such as the γ of the MDP or the choice of the ML framework that will be used, such as PyTorch, that can be found.

Once an instance of the algorithm class is created, some of the functions that can be done are *train*, *save*, or *compute_single_action*. The first one, as its name suggests, is used to determine the best policy, whilst *save* is used to preserve the algorithm's parameters so that they can be restored at a later time, including the trained policy. The *compute_single_action* function takes an environment observation as input and outputs a single action that will be delivered to the environment's *save* function in accordance with the most recent policy.

The checkpoint of the model is saved in a folder that also contains a file that TensorBoard can read by utilizing the *save* feature. TensorBoard is a tool created by the TensorFlow team that offers a visual platform to view or debug the training and is typically useful when working with machine learning algorithms. The TensorBoard Application is depicted in the figure 3.3

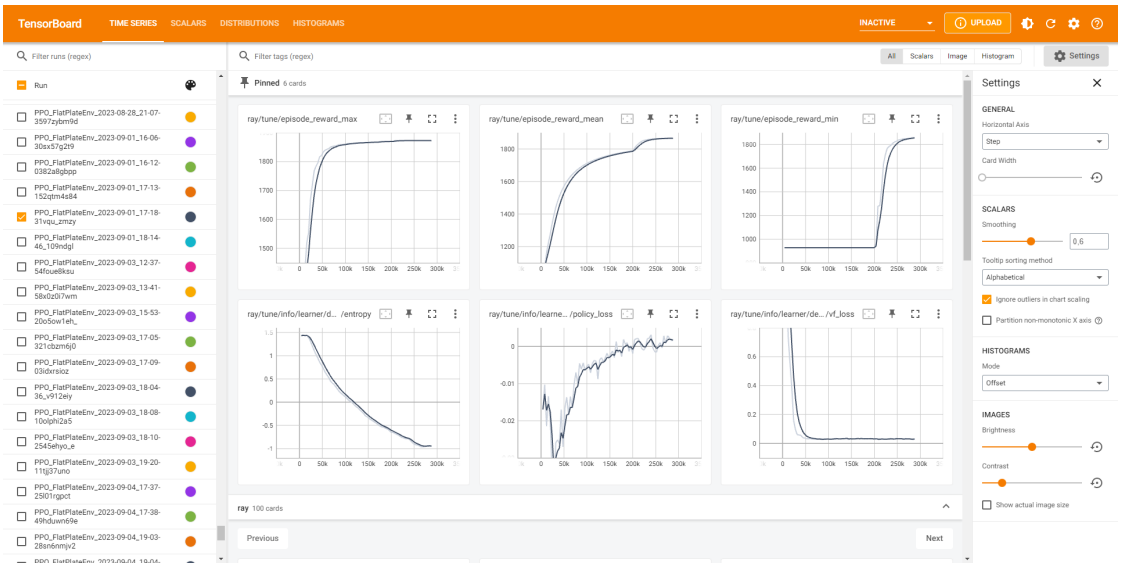

Figure 3.3. TensorBoard User Interface [13]

3.1.2 StarCCM+

"Simcenter STAR-CCM+ Software is a Computational Aided Engineering (CAE) solution for solving multidisciplinary problems in both fluid and solid continuum mechanics, within a single integrated user interface." [37] In StarCCM+ it is possible to import and create geometries, generate quality meshes, solve the governing equations, analyze the results, automate the simulation workflows for design explorations studies and even connect to other CAE software for co-simulations. The latest is of great importance to this project as it will allow the fluid simulation to be influenced by external Python software running the control algorithm based on AI. All of the functions mentioned before are conveniently integrated into the same interface, making the development of simulation really intuitive and procedural.

3.1.3 FMI standard

"The Functional Mock-up Interface (FMI) is a free standard that defines a container and an interface to exchange dynamic models using a combination of XML files, binaries and C code, distributed as a ZIP file." [18]

The development of this standard and the adoption of it by more than 170 tools, like StarCCM+, or Python, makes for a perfect environment to allow for collaborations between different simulations software. In this project, the integration between the AI model and the accurate CFD simulation is able to be carried out thanks to the development of this standard.

3.1.3.1 PythonFMU

The PythonFMU library [17] is the Python implementation of the FMI standard. It is "a lightweight framework that enables the packaging of Python 3 code or CSV files as co-simulation FMUs" [17], making it perfect for the integration of the AI control model with the CAE simulation software. The API requires the code to be defined inside a Python class with an *init* function in which the registration of variables is carried out following a well developed documentation. This function, as for all Python classes, is only executed when the class is instantiated and must also include all the code that we consider important to run first and just for once. The *do-step* function, on the other hand, will import the current time of the simulation and the time step, and it is executed at every step called by the parent simulation, which in this project will be the StarCCM+ CFD simulation. Any value declared as output in the *init* function will be sent to the parent simulation after the execution of the *do-step* function.

3.1.4 Flask

"Flask is a web framework [written in Python]. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website." [38]

Flask is a type of micro-framework, which normally have little to no dependencies to external libraries. This allows for a light framework, which is ideal for simple

web applications, but would produce a lot of work by yourself for more complex applications.

For the project, the use of Flask will be vital to develop an API running as a server, in which a client, by sending to a certain *http* direction the data they want to predict, will get in return the predicted data, i.e the control action. The usage of Flask in the project is just one part in order to simplify the inference of the AI model and separate it from other Python software, simplifying the software stack and interconnections, avoiding integration errors.

3.1.5 Docker

The usage of the Flask, as mentioned, is just part of the process to make the inference of the AI model independent. Docker completes this process by allowing to run the Flask server in an isolated environment in the same computer as other code.

"Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly." [39]

Docker is able to pack and run an application in an isolated environment called containers. The isolation and security allows to run many containers simultaneously on a given host. Containers are light weight and contain everything needed to run the application, not relying on installed software on the host, thus, making it such an interesting solution for this project. The usage of a Docker container to run the Flask application means that no conflicts with any Python application running on the host will happen.

This was necessary for the project, due to the incompatibility of StarCCM+ with the usage of Python virtual environments. Without containers, the AI development frameworks and the other frameworks used for the project would cause many conflicts between dependencies, making the project really complicated or imposible to complete.

3.1.6 Anaconda

Anyone who works with code, especially open-source languages like Python, may discover that when creating a program on top of various libraries that may be sup-

ported by various parties and those libraries may also have dependencies, the problem is carried down and has a cascading effect that can break a lot of code.

Utilizing virtual environments is good practice. It is possible to run different versions of the same libraries, and even of Python, on the same machine thanks to virtual environments, which separate copies of the Python interpreter and its libraries and guarantee that no version conflicts will result.

Anaconda is a fantastic tool for building virtual environments. "Anaconda is a package manager, an environment manager, a Python/R data science distribution and a collection of over 7500 open-source packages" [40]. Environment in this context refers to a virtual environment rather than a RL one.

Both the UI and the command prompt in Anaconda are user-friendly and make it easy to activate or deactivate environments or install packages. Given the significant number of dependencies for this project, not using a virtual environment is practically never an option, and using Anaconda will be quite helpful.

3.1.7 Writing code

Code can be written in many different kinds of environments. For the project two different philosophies of writing code are used that benefit different parts of the process.

3.1.8 Jupyter Notebooks

Notebooks execute code written in different cells that all run with the same Python Kernel. The ability to test multiple functions or sets of code independently allows for an extensive amount of code experimentation when using a notebook, especially when setting up the training API for RL algorithms, and building the different environments. Jupyter Notebooks is the notebook environment used for the project, it is open-source, easy to install and to use, and has a powerful environment, making it a popular choice in the data science community.

3.1.8.1 Visual Studio Code

Integrated Development Environments (IDE) are solutions for creating code. A source code editor, an interpreter or set of build tools, and a debugger are typically included.

By using this type of environment, more complete and fully functional code that runs in a command prompt rather than cell by cell can be written. The project's IDE is Visual Studio Code, a free and open-source program that makes use of the Python interpreter running in the Anaconda environment. The project uses the code developed in these types of environments to execute training in a command prompt with a script that handles everything in a single run.

3.1.8.2 Version control

Version control allows for new snippets of code or modification to already existing code to be correctly integrated avoiding bugs in the process. It is also a great way of having a history of modifications easily available. The code is usually hosted in an online hosting platform as a repository, that all the people working in the project can access and modify, but it can also allow for the general public to check the code, either for them to use, or even to propose changes to it. In this project, Git [41], and Github [42] are the choices for the version control software and the hosting platform respectively.

3.2 Configuration

As a remainder, the problem to solve will be a 2D flat plate with 2 degrees of freedom, one vertical w and another torsional θ. This section covers the description of how the problem is set up for solving using the software described at the beginning of the chapter. For this, it will first be necessary to create an environment that describes the behavior of the problem, then utilize the chosen techniques of control, and finally integrate the control with StarCCM+ for testing. It is important to note, the AI model is developed using a reduced order model allowing for faster computation times in a really heavy data requirement environment like it is ML model training. Then the model is verified in a better accuracy simulation carried out in StarCCM+, a state-of-the-art commercial CAE simulation software.

3.2.1 Environment

In this section, the set up of the specific environment for a 2D flat plate aeroelastic model compliant with the Gymnasium API is discussed.

3.2.1.1 *Init* function

As previously discussed, the *init* function runs only once when the environment is instantiated. The function takes as and input a Python dictionary with external configuration. This configuration will set the time-step to $2.5 \cdot 10^{-4}$ and the total steps to 2000 for each training episode, totalling to $0.5s$, and to 4000 during inference, totalling $1s$. In this function also, the constants that define the properties of the simulation are defined. Among these constants the air density $\rho_\infty = 1.18kg/m^3$ can be found, the chord length $c = 0.1m$, the thickness of the flat plate $h = 0.04c$, its length $l = 0.37$, the Young's modulus $E = 2 \cdot 10^9 N/m^2$, the Poisson coefficient $\nu = 0.35$, the inertia of the section $I = 3.9 \cdot 10^{-4} kgm^2/m$, the mass $m = 0.36kg/m$, the rigidity of the torsional degree of freedom $k_\theta = 36.46Nm/m$, and the one of the vertical degree of freedom $k_w = 558.46N/m^2$. On top of that, two important parts have to be added to comply with the Gymnasium API in order for the environment to correctly be accessed by the RLlib framework: the observation space, and the action space.

For the *observation space*, we take the higher and lower bounds that we expect the observation to be. This serves to throw errors in case a certain value boundary we know that our observation has has been violated, usually due to code errors and, thus, avoiding numerical errors in the training. The observation for this environment is a vector of 5 numbers, and they're all bounded between the same values -1000 and 1000 to avoid code errors interrupting the training.

The *action space* bounds the action of the policy. In the problem to solve this is just a single value that will correspond to a torsional moment that will be used to control. This value's bounds can be approximated, as the exact value is out of the scope of the project and would correspond to a further research. To approximate this value we can see what the moment coefficient of a flap attached at 3/4 of the chord of the flat plate would be.

$$cm_{LE} = -\frac{\pi}{2}\alpha - \frac{1}{2}\eta\left(\frac{5\sqrt{3}}{4}\frac{\pi}{3}\right) \approx -\frac{\pi}{2}\alpha - 1.61\eta \tag{63}$$

The equation (63) [43] shows the moment coefficient for a flat plate with a flap at 3/4 of the chord given the angle of attack α and the angle of flap deflection η. Knowing the equation (64), and approximated maximum deflection angle of 20°, the boundary of $1.14[N \cdot m]$ can be obtained, but for simplicity and to be conservative, $1[N \cdot m]$

will be used. For this $\rho_\infty = 1.18kg/m^3$, $c = 0.1$ and $V_\infty \approx 18$. V_∞ depends on the adimensional rigidity $k^* = k_\theta / \frac{1}{2}\rho_\infty V_\infty^2 c^2$, and as it serves only as an approximation, to train we take the one for the value $k^* = 15$, which should provide with a lot of oscillation to the flat plate.

$$M = \frac{1}{2}\rho_\infty c^2 V_\infty^2 cm \tag{64}$$

$$k^* = \frac{k_\theta}{\frac{1}{2}\rho_\infty V_\infty^2 c^2} \tag{65}$$

3.2.1.2 *Reset* function

In the reset function all the arrays are initialized as empty, the $k^* = 15$ parameter is created and the V_∞ is calculated isolating it from the k^* expression presented in the equation (65).

The initial AOA is set to be 2.5 and a few iterations of calculations fixing this value as to establish an average AOA of 2.5 when the aeroelastic calculation starts. As there's not structural deformation when fixing the AOA, these values are not calculated, only aerodynamic values, that will also be calculated in the step function and, thus, better explained there.

3.2.1.3 *Step* function

As previously mentioned, the step function advanced one time step every time it is called. When the flag variable *terminated* is set to *True* the automated experience collection finishes. These will happen if the AOA value goes over 60 degrees and when the maximum steps we have configured the environment to run are reached. Also, this equation takes as the input the control moment M_{ctrl}.

In this section the aeroelastic equations for the torsional and vertical degrees of freedom, shown in the equation (3.2.1.3) and equation (3.2.1.3) respectively, and developed in Chapter 2, are solved using the *Runge-Kutta* order 4 method for the first steps, initialization, and *Predictor Corrector* method for the rest of the episode.

$$I\ddot{\theta} + k_\theta\theta = \frac{1}{2}\rho_\infty c^2 V_\infty^2 \left(c_m^{st}(\alpha, w) + c_m^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha})\right) \longrightarrow$$
$$I^*\ddot{\theta}^* + k^*\theta^* = c_m^{st}(\alpha, w) + c_m^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha})$$

$$m\ddot{w} + k_w w = \frac{1}{2}\rho_\infty c^2 V_\infty^2 \left(c_l^{st}(\alpha, w) + c_l^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha})\right) \longrightarrow$$
$$m^*\ddot{w}^* + k_w^* w^* = c_l^{st}(\alpha, w) + c_l^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha})$$

The c_l^{st} and the c_m^{st} are obtained from the polar of the flat plate that has been calculated using StarCCM+

The c_l^{dyn} and the c_m^{dyn} are outputs of a NN that takes the arguments $\overline{\alpha}$, $\Delta\alpha$, $\dot{\alpha}$, $\ddot{\alpha}$ to predict these coefficients and that was trained in Torregrosa et al., 2021 [14].

The added moment M_{ctrl} is introduced in the torsional degree if freedom equation resulting in the equation (66).

$$I_{2D}\ddot{\theta} + k_\theta\theta = \frac{1}{2}\rho_\infty c^2 V_\infty^2 \left(c_m^{st}(\alpha, w) + c_m^{dyn}(\overline{\alpha}, \Delta\alpha, \dot{\alpha}, \ddot{\alpha})\right) + M_{ctrl} \tag{66}$$

This function also outputs an array of observations containing the current AOA α_i, the derivative of the AOA $\alpha_i - \alpha_{i-1}$, the initial/reference AOA α_0, the absolute error between the reference AOA and the current one $|\alpha_0 - \alpha_i|$ and the normalized value of the last input moment $M_{ctrl_i}/1.12$.

The reward has to be returned by this function as well. Which behaves like shown in algorithm 2.

Algorithm 2: Reward function logic

Get *error* and initialize reward R:

$R \leftarrow 0$

$error \leftarrow= \alpha_0 - \alpha_i$

if $|error| == 0$ **then**

| $R = R + 0.01 \rightarrow$ *Promotes having* 0 *error*

end

$R = R + (1 - |error|/3.5) \rightarrow$ *Guides the learning towards* 0 *error*

$R = R + 0.01(-|M_{ctrl_i}|/1.12) \rightarrow$ *Minimizes the amplitude of moment applied*

3.2.2 Classic control - PID

As a way to compare performance, it is interesting to also evaluate a classic control algorithm like the PID and, for that reason, a function can be coded and used to interact with the environment programmed as described above. The algorithm 3 shows the behavior of the controller.

Algorithm 3: PID controller logic

Set gains:

$K_u = 8$

$t_u = 0.00875$

$K_p = 0.6 \cdot ku$

$K_i = 1.2 \cdot ku/tu$

$K_d = 0.075 \cdot ku \cdot tu$

$I = k_i \cdot error \cdot \Delta t$

$D = (error - error_{t-1})/\Delta t$

$M_{ctrl} = k_p \cdot error + k_d \cdot D + I$

The PID gains were tuned using the Ziegler Nichols procedure [44]. Where k_u is the critical gain and t_u the oscillation period.

3.2.3 Reinforcement learning

To design a RL based control algorithm, as mentioned in previous sections, the RLlib API is an easy option to use. The choice of the algorithm is PPO and the hyperparameters chosen are $\gamma = 0.95$, kl_coeff, $\lambda = 0.9$, $entropy_coeff = 0.01$, and $lr = 0.0001$ which will decrease to $lr = 0.00001$ after 60000 training iterations for convergence purposes. The training data, which will correspond to the observations and the reward returned by the *step* function, will pass in batches of 128 which helps with memory management.

3.2.4 StarCCM+ integration

The development of the RL algorithms and all their related tasks like environment construction have been done using Anaconda virtual environments. However, StarCCM+ is not design to work with environments and requires Python to be added to PATH in Windows. For this, it is necessary to run the RL model in a Flask REST

API that is later *dockerized* to run as a separate container inside windows, allowing us to execute Python from PATH without interfering with the Python running the control algorithm.

The REST API runs in a local port of the Docker container, in this case port 5000 and contains a method called *predict* which gets passed the input data necessary for it to predict in a JSON format through "http://localhost:5000/predict".

3.2.4.1 FMU code

Following the PythonFMU API, the variables we want to use in the code have to be declared in order for StarCCM+ to recognize them inside its interface. The input to the FMU will solely be the AOA coming from StarCCM+, the output will be the control moment M_{ctrl}. Also, some local variables related to the processing of the input data, i.e observations, are defined to be tracked inside the StarCCM+ interface.

A Python dictionary is defined containing the inputs that we want to pass to the RL algorithm i.e the observations defined in the environment *step* function.

This dictionary is then passed in a JSON format to the *predict* method of the REST API running the RL model, which returns a JSON format response containing the action moment M_{crtl} that is then extracted and passed as the output of the FMU.

The Python code has to be compiled to an FMU model using PythonFMU, which prepares the model to be recognized by StarCCM+ and get executed.

3.2.4.2 CFD simulation

The CFD simulation is obtained from Gil et al., 2021 [45]. The spring combination results from the model defined in Chapter 2, the initial AOA is 2.5, as mentioned before, resulting in the diagram of the figure 3.4.

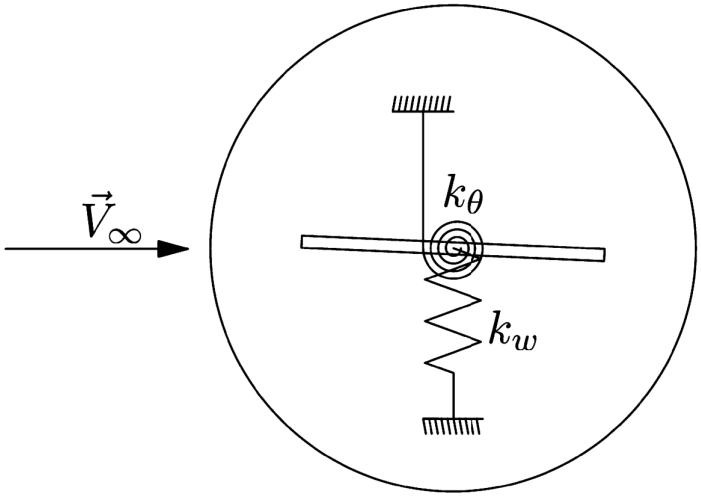

Figure 3.4. Sketch of the spring set-up applied to the overset mesh as defined in Torregrosa et. al., 2021 [14]

Also, the domain is created following a set of dimensions depending on the chord length, being the upstream length $L_u = 5c$, the downstream length $L_d = 15c$ and the width $H = 4c$. The upstream and downstream lengths are chosen in order for the inlet and outlet boundary conditions to not affect the simulation. The domain can be seen in the figure 3.5.

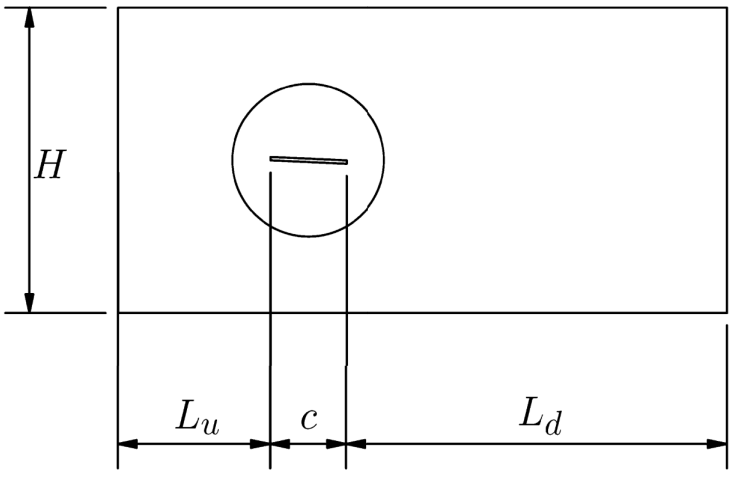

Figure 3.5. 2D domain (not to scale) as defined in Torregrosa et. al., 2021 [14]

The torsional and vertical displacements are calculated using the Finite Volume Method inside StarCCM+, solving the Unsteady Reynolds Average Navier Stokes *URANS* equations for the fluid flow and the rigid solid motion equations for the plate, using the same time-step used for the environment configuration from a past

section $2.5 \cdot 10^{-}4s$. To be able to calculate flow separation under adverse pressure gradients the turbulence model $k - \omega$ with sheer stress transport SST is chosen. A coupled solver with second order upwind Roe Flux Difference Splitting scheme for the advection terms is used. The gradients get computed using a hybrid between Gauss and Least Squares method with Venkatakrishnan limiter. For the transient simulations second order time discretization is used.

For the discretized computational domain, an overset mesh is used. The size of the mesh at the wall of the plate is approximately 0.004, gradually increasing size until a uniform overset domain size of 0.02, this way it is ensured that the interface is similar sizes at the overset and background domains. If the flat plate happens to be at a medium-high angle of attack, specially, an important wake will be formed for which downstream the plate the size of the cell is constrained to 0.040. The biggest size of the mesh is set to 0.200. For the boundary layer, a prism layer is created near the wall of the plate with a thickness of 0.075 and 5 layers, making sure the $y+ < 1$ for the most part of the plate wall. The mesh with this configuration results in approximately $5.1 \cdot 10^4$, and has been chosen making a mesh independence method based on Richardson's extrapolation RE, the resulting mesh can be seen in the figure 3.6.

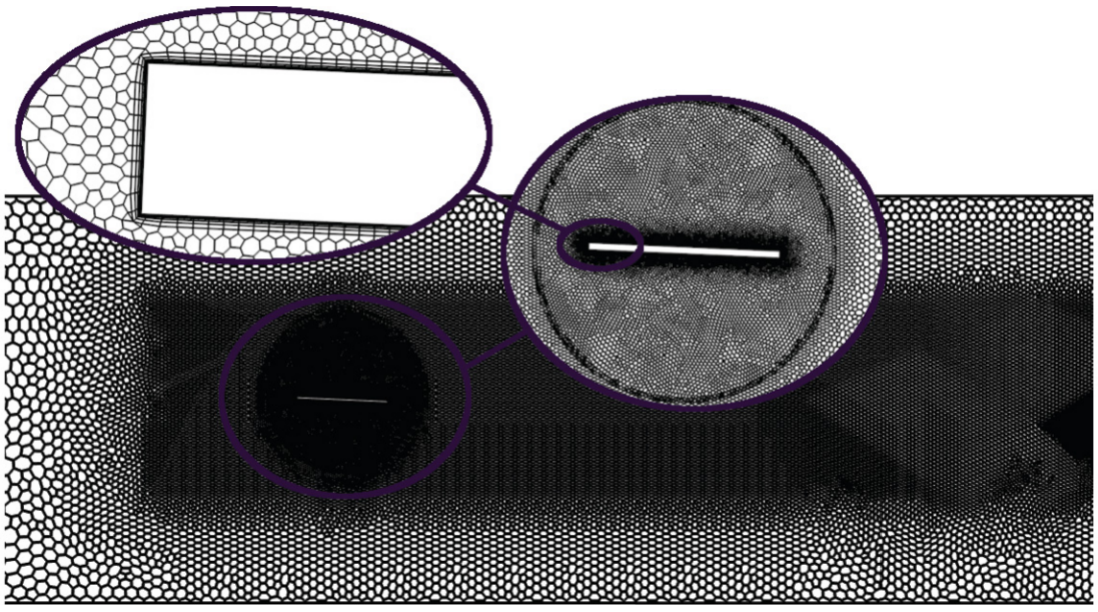

Figure 3.6. Sketch of the computational mesh as defined in Torregrosa et. al., 2021 [14]

The simulation has to perform what StarCCM+ literature calls co-simulation using the FMI standard. For this, the simulation has to be further configured to

allow it. The first step is to activate in the fluid continuum the optional co-simulation model with the FMI option. The external physics that will be calculated require a of a new physics continuum, with the optional external continuum activated, with a network topology space, FMI external application and for this project, as mentioned before, implicit unsteady time. The continuum that contains the FMI physics has to be assigned to a new region, that can be called FMU. With the new node called external links, the new link that was created has to be assigned an external continua corresponding to FMU and the coupling negotiation option of time-step, which means after every simulation time-step, StarCCM+ will communicate with the external FMU.

After the previous configuration steps are performed, it is necessary to import the FMU library that corresponds to the control model developed in Python, with the import user library option in StarCCM+. After being imported, in the FMI resource reference from the link options, the FMI library will be set to the imported library.

The exchanged values between the RL model and the StarCCM+ values have to be configured as well. In the link node, the option import settings will populate the tree with the values that are defined in the FMU model. The FMU will need the AOA of the flat plate, for which a report is created that calculates it from the vertical velocity of the plate and its torsional angle and is sent to the corresponding value in the link tree. For the imported values, the moment is added as an external moment in the node that corresponds to the structural physics of the plate.

Chapter 4

Results

In the following pages the results are presented, comparing the different control methods. As commented in previous sections, the RL algorithm is trained with a $k^* = 15$. For this, it is expected for the algorithm to perform the best for this value of k^*. However, out of interest, the response to higher values of V_∞ can be tested, lowering the value of k^*.

4.1 No control

4.1.1 $k^* = 15$

It is interesting to know how both models' response perform without any kind of control. For the reference $k^* = 15$, used for training, it is shown in the figure 4.1. The AOA is calculated from the vertical velocity $\dot{w}$ and the torsion angle θ as seen in the equation (67). In the figure 4.1 this relationship can clearly be observed as the AOA response seems to be composed of two frequencies combined, a higher frequency corresponding to the θ degree of freedom, and a lower frequency corresponding to the w degree of freedom.

$$\alpha \approx \theta - \dot{w}/V_\infty [rad] \tag{67}$$

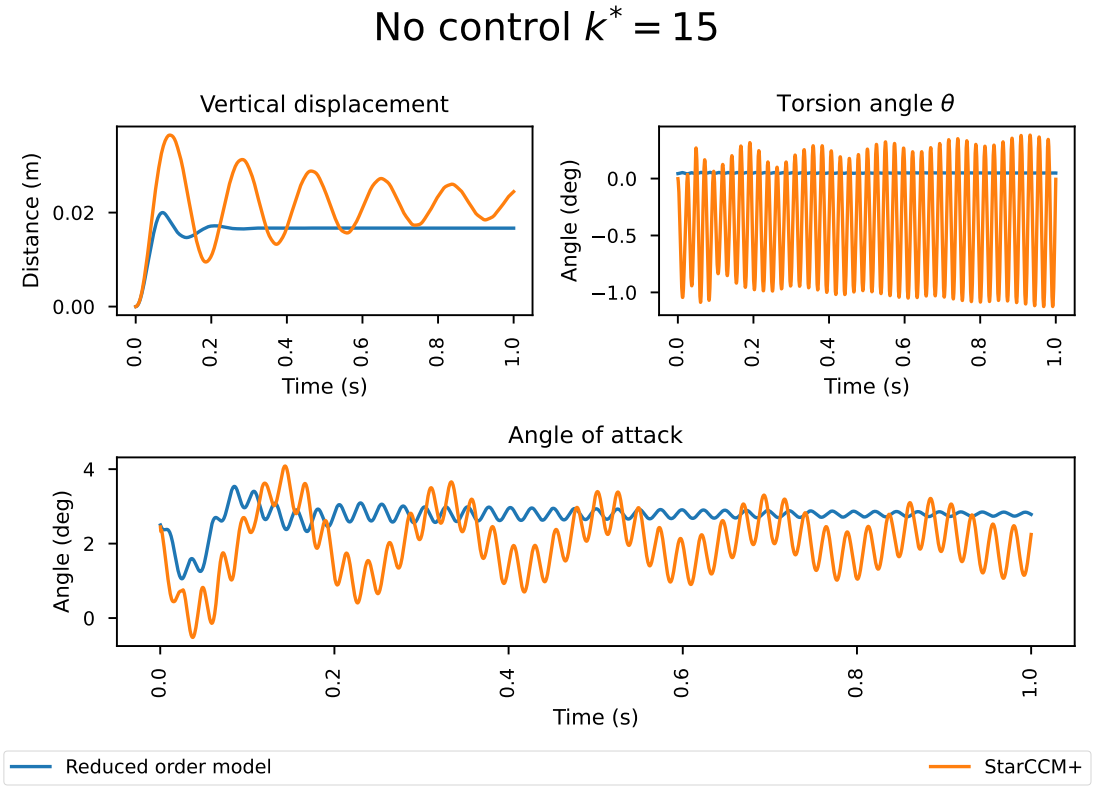

Figure 4.1. Angle of attack, torsion and vertical displacement for a $k^* = 15$ and initial angle of attack of $2.5(deg)$ in StarCCM+ and reduced order model.

A substantial difference can be seen in the response of both models. The reduced order model dynamic part of the aerodynamic coefficients $c_{l,m}^{dyn}$ are obtained using a NN trained with $\overline{\alpha}$, $\Delta\alpha$, $\dot{\alpha}$ and $\ddot{\alpha}$ and performs pretty well, however some effects are lost, for instance the fluctuations of the wake, or vortex shredding, are not reproduced. The velocity scenes extracted from the StarCCM+ simulation and shown in the figure 4.2 clearly show the appearance of vortexes in the wake. For the control of the aeroelastic phenomena, as the idea is to maintain a certain AOA, the error in training created by the reduced order model is expected to affect less than for a free motion, considering the oscillations will be of lower amplitude.

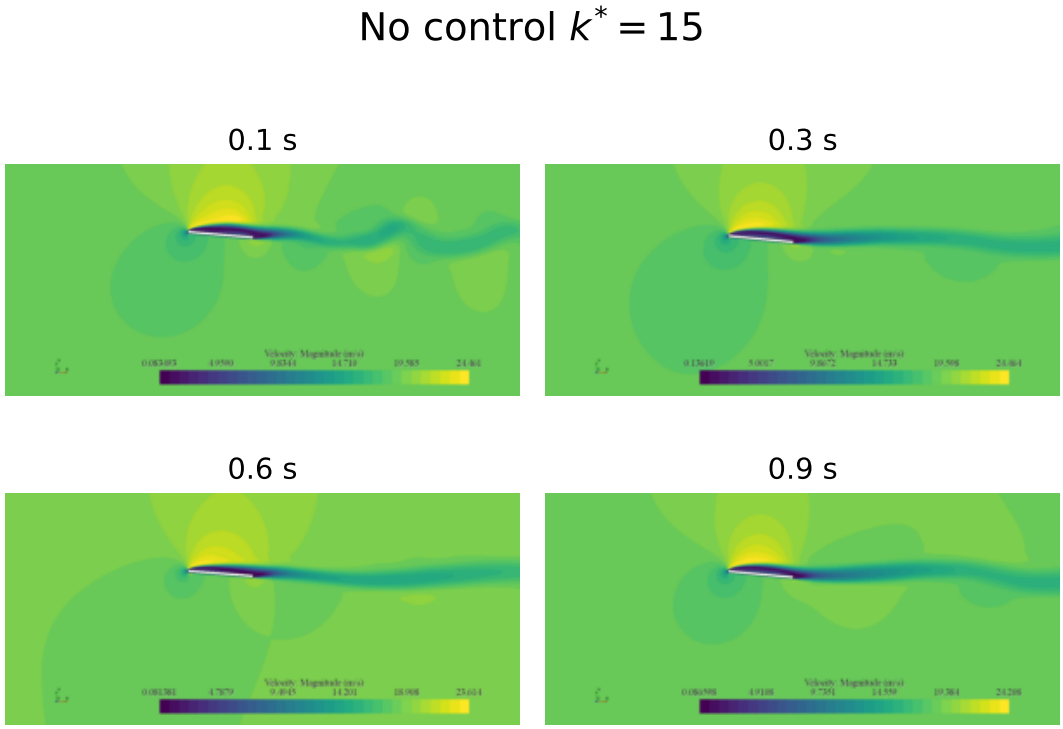

Figure 4.2. Velocity contours for a $k^* = 15$ and initial angle of attack of 2.5(deg) in StarCCM+ for 4 for times 0.1s, 0.3s, 0.6s, and 0.9s

4.1.2 $k^* = 9$

As the idea is to also test a higher velocity case. The response to a lower $k^* = 9$ is also computed as a comparison to the training case. This can be seen in the figure 4.3. As it can be observed, the response has a much larger amplitude. The vertical displacement w oscillates and then stabilizes, but with a high vibration, with similar frequency to the torsion θ oscillations. In the period represented, θ continuously rises and oscillates with a higher amplitude than $5deg$.

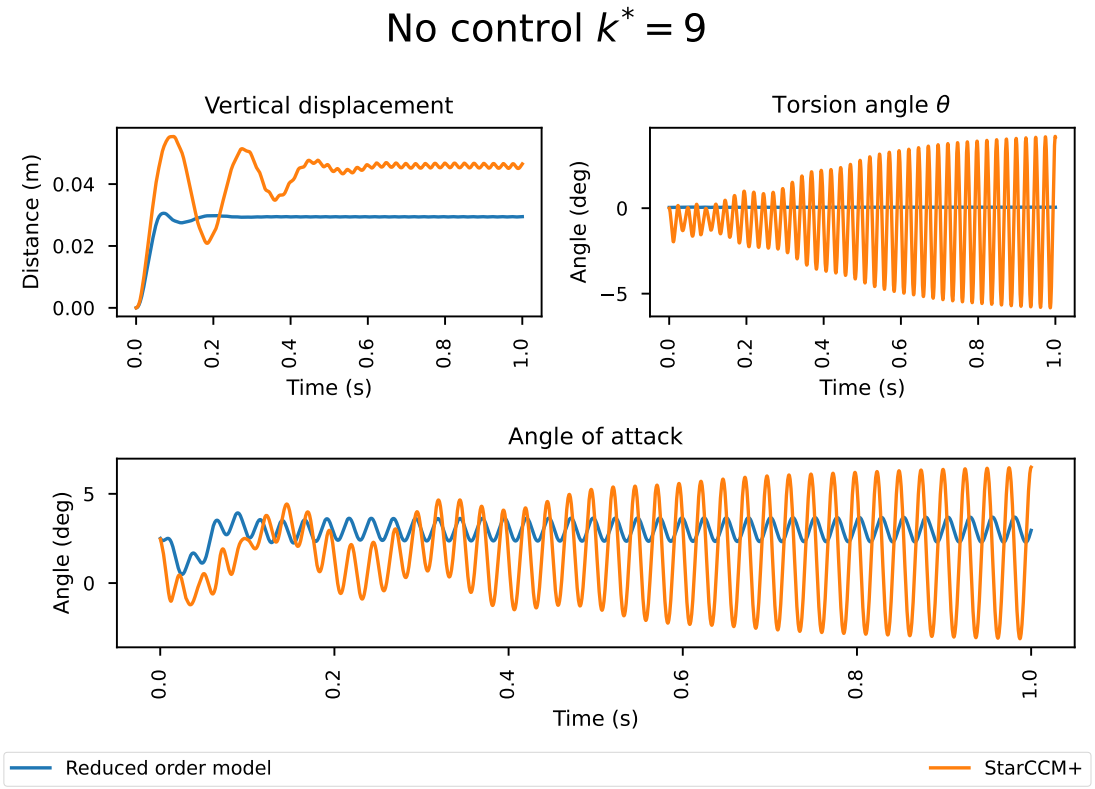

Figure 4.3. Angle of attack, torsion and vertical displacement for a $k^* = 9$ and initial angle of attack of $2.5(deg)$ in StarCCM+ and reduced order model.

In the figure 4.4, the vortexes that are created with the flat plate oscillating are, of course much higher. Which, in terms of control, will definitely suppose a bigger challenge.

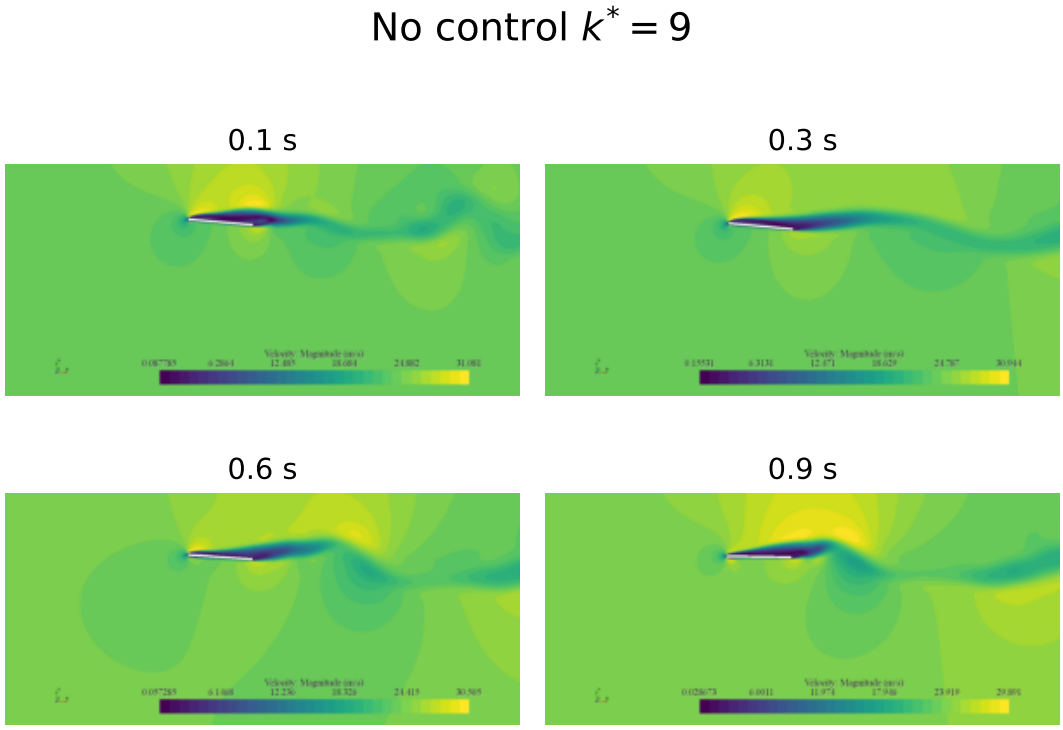

Figure 4.4. Velocity contours for a $k^* = 9$ and initial angle of attack of $2.5(deg)$ in StarCCM+ for times $0.1s$, $0.3s$, $0.6s$, and $0.9s$

4.2 Classic control - PID

For testing purposes it is interesting to evaluate the performance of classic control algorithms and see what difference they present in terms of results compared to a RL based control. In the figure 4.5 the response to a PID control is observed between both the reduced order model and the simulation in StarCCM+. The main difference we can see is that for the reduced order model, the response is a quite stable sinusoidal shape centered at the reference AOA. However, for StarCCM+, the response shape is noisier at the beginning and much better stabilized from 0.2 seconds and forwards. The control moment M_{ctrl} applied to the torsional degree of freedom is observed. As it could be guessed based on the AOA response, the reduced order model response is also a sinusoidal shape, with a different phase compared to the AOA, of course. For the StarCCM+ response, the action amplitude is quite smaller when stabilized and, as with the AOA case, it also presents a similar shape to that of the AOA, with a noisier transient that stabilizes past 0.2 seconds. It is necessary to highlight that the moment applicable by the PID has not been limited, and it can be seen that a big spike passes the boundaries. The vertical degree of freedom w and the torsional degree of freedom θ can also be observed in the figure 4.5. Continuing with the tendency observed for the AOA, the vertical displacement w for the StarCCM+ simulation reaches a steady slowly decaying its amplitude. The reduced order model response, however, seems to have quite more amplitude and it does not decay quite as fast, inducing a much bigger oscillation to the AOA. The θ response for the reduced order model shows a pretty flat sinusoidal shape that doesn't seem to decay, this hints that the majority of the AOA oscillation comes from the vertical degree of freedom w that it's not being directly controlled by the PID. The StarCCM+ response has a decaying shape that ends up stabilizing. It its interesting to note that despite the controlled degree of freedom θ seems to be responding a bit worse, the vertical displacement w contribution makes the StarCCM+ simulation stabilize better. It also interesting to point out that the vertical displacement barely has any noise, but the torsion angle has a much noisier response at the transient. This can also hint that the noisy AOA transient response can be heavily influenced by this degree of freedom, θ.

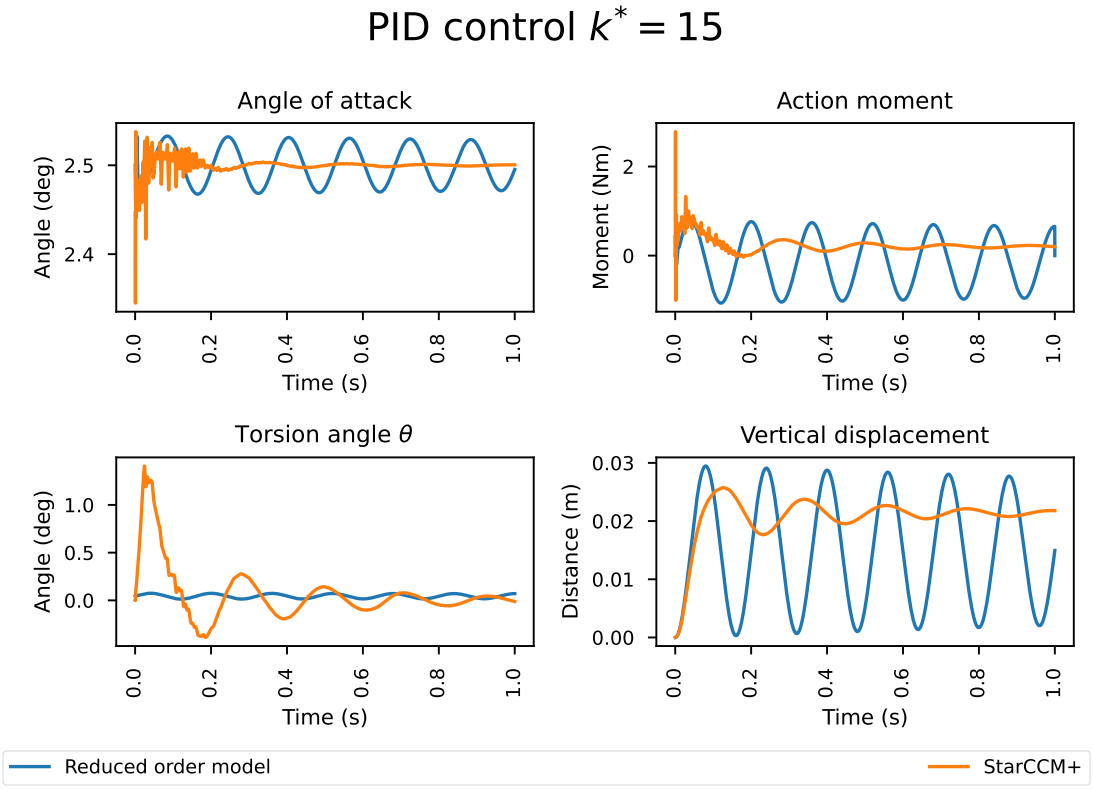

Figure 4.5. Angle of attack, torsion, vertical displacement, and action response from controlled case with PID for a $k^* = 15$ and initial angle of attack of $2.5(deg)$ in StarCCM+ and reduced order model

From the figure 4.6, it can be seen that the vortexes that appeared for $k^* = 15$ without control, are eventually eliminated, having a pretty steady flow. Proving, then, that the PID correctly controls the flat plate oscillation. However, a problem PIDs present, is that they are tuned for an specific set point. In this case, the set point is the one corresponding to $k^* = 15$, meaning that for different values, the controller will not work as well.

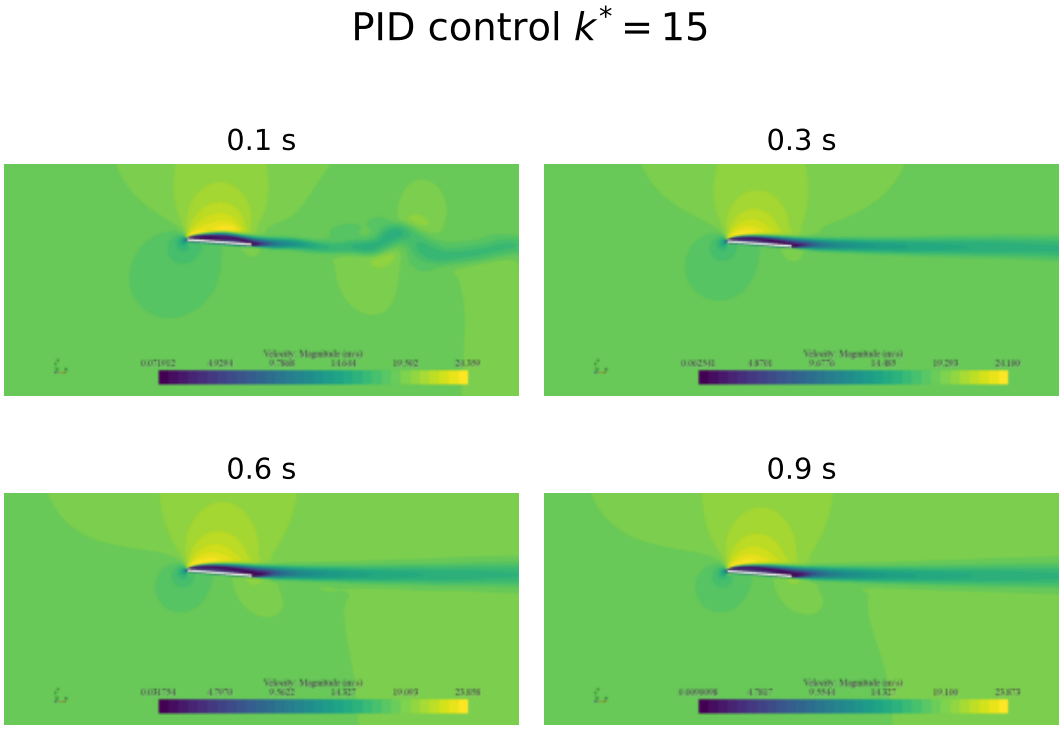

Figure 4.6. Velocity contours for a $k^* = 15$ controlled with PID with initial angle of attack of $2.5(deg)$ in StarCCM+ for times $0.1s$, $0.3s$, $0.6s$, and $0.9s$

4.3 Reinforcement learning

Now that the PID response has been evaluated, and the behavior is understood. The RL approach can be compared.

4.3.1 $k^* = 15$

In the figure 4.7, the AOA response can be seen for the RL based control case of $k^* = 15$. First issue that comes to notice is that now the responses are noisy. The reason behind this noise could be the statistical nature of the NNs, and it would be interesting to research how this noise could be reduced in training. The output from the reduced order model has a similar shape to the one from the PID control but with bigger amplitude and the commented added noise. For the StarCCM+ case, there is also a little amount of steady state error, the amplitude of the responses is also bigger, and the noise is also higher. The control moment M_{ctrl} applied by the RL based control has the overall behavior of the PID case, but seems to also have the commented added noise. The reduced order model now saturates the action,

probably because of the noise. Also, the StarCCM+ response now does not have the spike, and in that place it saturates, but now there is an imposed limit to the action provided by the controller. It can be seen that the vertical displacement w behaves almost the same as for the PID, although the reduced order model response has a bit less amplitude. It is important to note that there is no visible noise. That means that the noise comes from the θ degree of freedom, which is the controlled one, and undoubtedly puts the blame of the noise to the RL based control. Apart from that, both behave quite similarly, as it could be expected given the similarities in the AOA response and the control inputs.

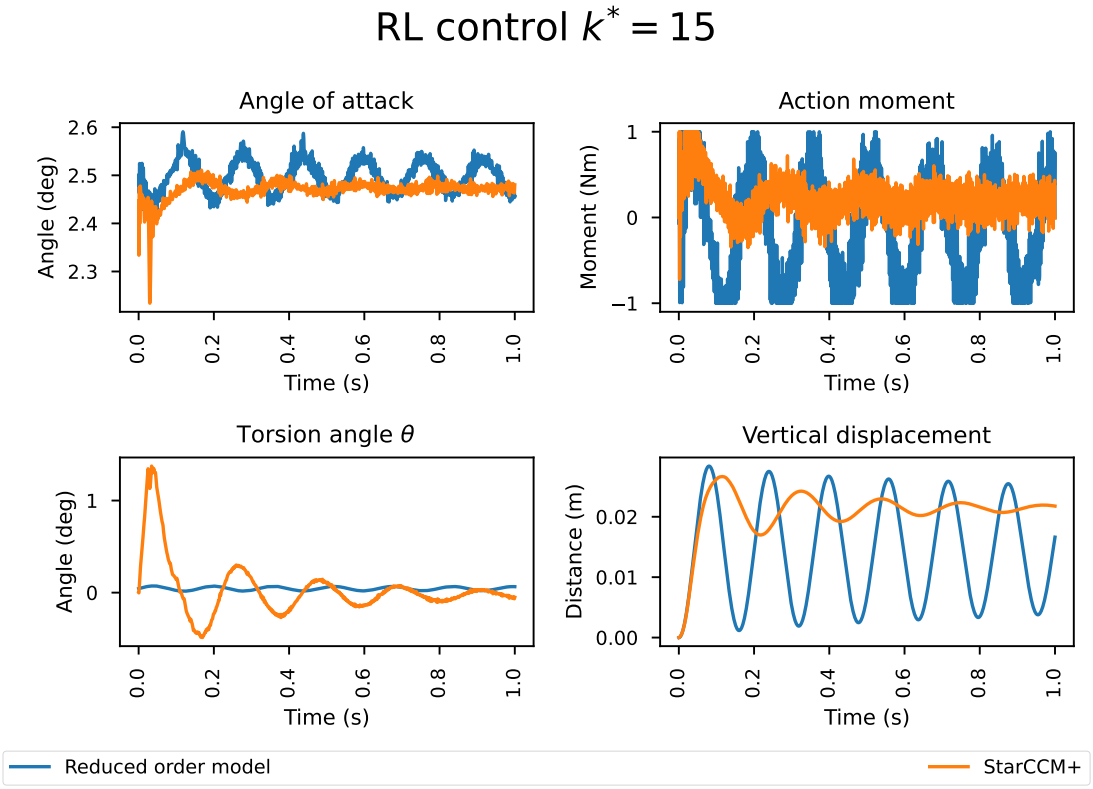

Figure 4.7. Angle of attack, torsion, vertical displacement and action for a $k^* = 15$ and initial angle of attack of $2.5(deg)$ in StarCCM+ and reduced order model.

The little oscillations that are created by the noise have their respective effect in the wake behind the flat plate, which can be seen in the figure 4.8, specially at times $0.6s$ and $0.9s$.

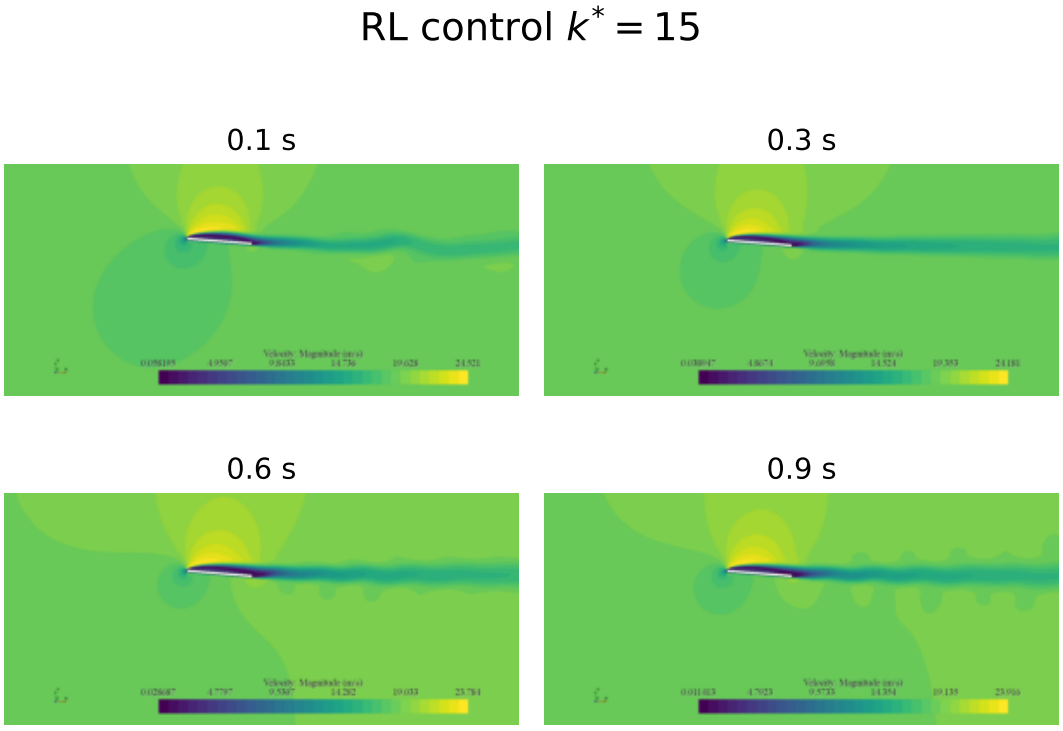

Figure 4.8. Velocity contours for a $k^* = 15$ controlled with RL with initial angle of attack of $2.5(deg)$ in StarCCM+ for times $0.1s$, $0.3s$, $0.6s$, and $0.9s$

4.3.2 $k^* = 9$

If the $k^* = 9$ case is tested, the response can be seen in the figure 4.9. For ease of visualization given the noise, the difficult to see responses have been added some transparency. The overall response is similar to the $k^* = 15$ with a few new features and a lot more added noise. The noise could be given the statistical noise produced by out-of-normal observations that were not seen during training, making the policy behave with less certainty i.e more entropy. The big oscillation from the action makes sense given that for a higher V_∞, the moment required to control pitch will increase, and right now it is limited. A new feature that is really interesting to see is a big spike in the AOA response for the reduced order model that is not present in the StarCCM+ response, it is interesting to see given the w and θ responses apparently do not give any clue to why that behavior is produced. Another spike that is of interest is for torsion θ in the StarCCM+ response, that is not present in the reduced order model, however this spike clearly produces an effect in the AOA response.

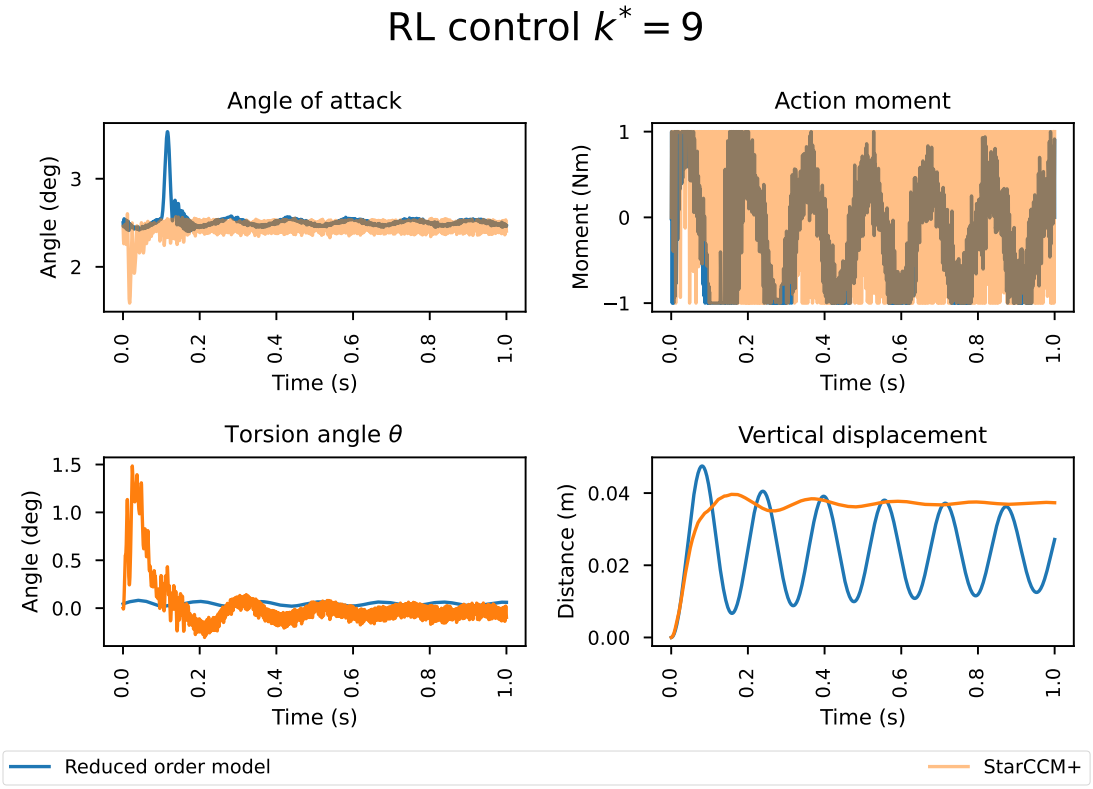

Figure 4.9. Angle of attack, torsion, vertical displacement and action for a $k^* = 9$ and initial angle of attack of $2.5(deg)$ in StarCCM+ and reduced order model.

The high vibrations present in this simulation have a clear effect on the wake, as it can be seen in the figure 4.10. Similarly to the $k^* = 15$ case, there is presence of a lot of wake turbulence produced by the vibration of the flat plate. In this scenario the effect is pretty drastic, but it is important to note that the non-controlled case for this k^* the flat plate oscillated heavily so, even though the vibrations a too drastic, the improvement is noticeable.

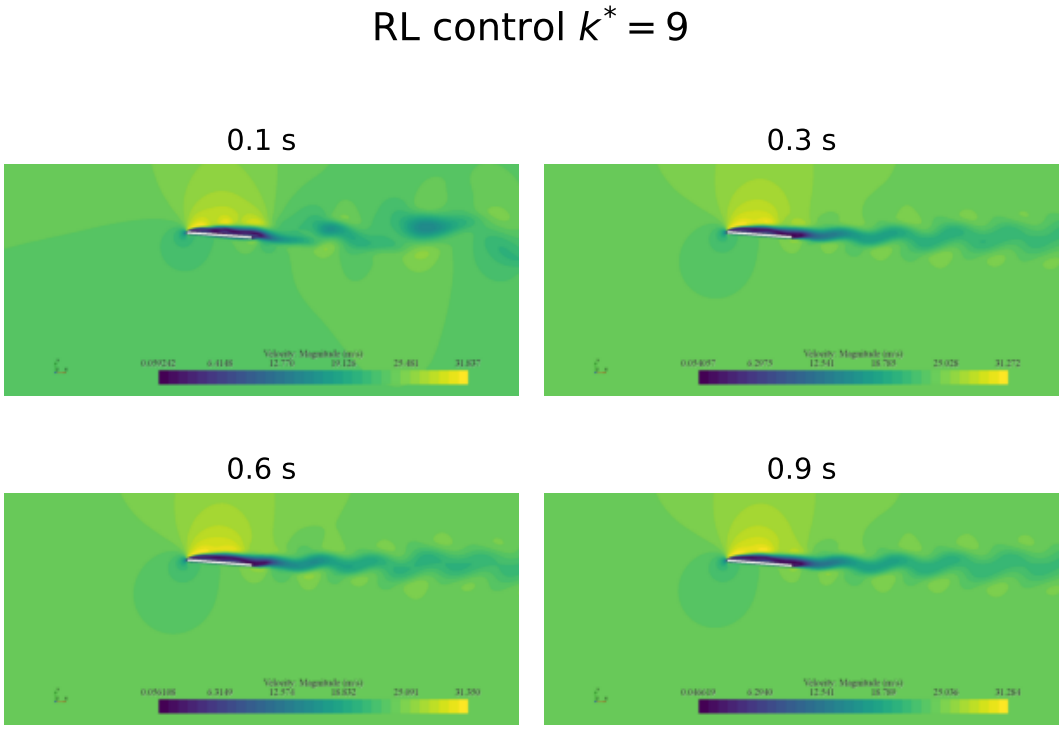

Figure 4.10. Velocity contours for a $k^* = 9$ controlled with RL with initial angle of attack of $2.5(deg)$ in StarCCM+ for times $0.1s$, $0.3s$, $0.6s$, and $0.9s$

Chapter 5

Conclusion

The project aimed at creating a software stack that allowed interaction between Python code and StarCCM+ simulations, allowing to test the possibility of using data based solutions to control a basic aeroelastic problem and comparing it to classic control solutions. After the realization of the project, it can be concluded that both the software stack works and that using data based control for aeroelastic problems is a feasible task. An important issue that it is raised, however, is the necessity of owning a reduced order model that makes the training of the NN policy doable in an average computer, which for more complex tasks could be a real challenge to overcome. Another issue that is raised is the necessity of dealing with the noise that the data based solution is presented with. It is important to note that all the simulations have been developed with research in mind and, thus, the simple nature of the problem, with the added lack of validation of the results, presents the project as a base case from which to advance into more accurate simulations and, maybe, experimental results, that make this technology more convincing in order to be used in a practical scenario.

Chapter 6

Future Work

From the conclusion, it is easy to identify several ideas that could be developed from this project.

6.1 Noise reduction

A key aspect that was discovered when analysing the results was the added noise that came with the usage of NNs. This could be a first challenge to overcome and that would drastically benefit the usage of this technique for control purposes.

6.2 Adaptive control

The idea of using data based solutions gives the opportunity to develop more general control solutions that cover a wide range of flight regimes and initial conditions. Is is something into what this project has not made any conclusions, as it has just focused on a 2.5 degree angle of attack, and the same adimensional rigidity k^* for training. It would be a good idea that could be put into place and tested quite simply. A challenge to overcome would be the necessity of more training time, considering the wider range of outcomes the new environment would have given the constant change of physical conditions that would need to be done in order to properly generalize the training.

6.3 Test more algorithms

The project has focused on using the PPO algorithm to train the policy that controls the flat plate. This does not mean that it is the best algorithm to do so, but it is a good starting point given its flexibility and its ease for converging. A good idea would be to use more algorithms that could potentially make the training more effective. Also, PPO is an on-policy algorithm, so it would be interesting to observe what happens an off-policy algorithm is used. The great amount of options could provide with really interesting solutions that have yet to be explored.

6.4 Adapt to sensor, actuator constraints and other real life conditions

Before even trying to think about a potential real test or application that could be performed using this technique, it is necessary to develop the algorithm to take into account many factors that have been ignored in this project for simplicity purposes. Among these many factors sensor acquisition rate is an important one to take into account, but also the dynamics of the actuator. This project has applied a control moment directly to the flat plate in order to stabilize it, which leads to think that a flap could be a good solution. However, how the actuator is going to perform and the different aerodynamic phenomena that could make the implementation of a flap as a mean to control the flat plate feasible are yet to be tested. The real time performance will also be a task to address.

6.4.1 The Problem of High Computing Requirements

If a simple problem like this can take hours to compute. Is it going to be adequate to control more complex problems?. Also, inference of small NNs is quite fast but, can it be fast enough to be able to perform in real time applications? These are questions that have not been touched during the development of the project and that can be critical to the future of this technology as a control algorithm.

6.5 Experimental verification

A true test that could prove this technology's potential to control aeroelastic phenomena would need to solve all the issues presented previously in this chapter and develop a test rig for a wind tunnel that could provide with the experimental data that verifies its potential. For that, computing time will be essential, actuator and sensor modelling will be key, and a well refined model will be necessary to handle the experiment the best way possible.

Chapter 7

Budget

In this chapter the budget required for the realization of the project is discussed. It will also include a final cost breakdown.

7.1 Salaries

The first thing to determine is the cost per hour worked. The average salary of an aerospace engineer in Spain is 24.10 €/h.

The project had a duration of around 7 months and the total amount of hours worked is estimated at an average of 30 hours. This makes the amount of hours dedicated by the student to be:

$$30\ \frac{hours}{week}\ x\ 4\ \frac{weeks}{month}\ x\ 7\ months\ =\ 840\ h \tag{68}$$

The salary costs result in what is shown at table 7.1.

Concept	Quantity (h)	Cost per unit (€)	Total cost (€)
Salary expense of tutor	42	32	1344
Salary expense of engineer	840	24.10	20244
Total salaries			21588

Table 7.1. Salaries

7.2 Equipment and Software

The cost of equipment is going to take into account the amortization of the equipment used and the software licenses.

The cost of the equipment used taking into the account the amortization can be seen in table 7.2.

Equipment	Amortization period	Total cost (€)	Utilization period	Cost in the project (€)
ASUS ROG Strix G712LV	48 months	1499	7 months	218.60

Table 7.2. Equipment cost

The software used in thee project is all open-source, except StarCCM+. The software has been mentioned in a previous chapter. Usage of StarCCM+ has been made using their PoD licenses that amount to 24 €/h. With an approximate 30 hour usage, it sums up to 720€

7.3 Indirect costs

Indirect costs are the ones that do not depend directly on the project, but the fact that the project is done originates this costs. Indirect costs are the ones related to the job done by the administration of the school, power consumption and internet. These cost are hard to calculate due to the complexity and variety of them so it is approximated to 5% of the total budget of the project. Indirect costs are shown in table 7.3

Concept	Total	Percentage (€)	Total cost (€)
Indirect Cost	21806.6	5%	1090.33

Table 7.3. Indirect costs

The total expenses originated by the project divided by category and the total cost is shown in table 7.4.

Concept	Total (€)
Salary	21588
Equipment	218.60
Software	720
Indirect	1090.33
Final	23616.93

Table 7.4. Final budget

Bibliography

[1] M. Oduncuoglu and H. I. Kurt, *The basic structure of an artificial neuron*, May 2015. [Online]. Available: https://www.researchgate.net/figure/The-basic-structure-of-an-artificial-neuron-43_fig3_282849515

[2] Totorialspoint. A typical ann. [Online]. Available: https://www.tutorialspoint.com/artificial_intelligence/images/atypical_ann.jpg

[3] OpenNN. Deep neural network. [Online]. Available: https://www.neuraldesigner.com/images/deep_neural_network_big.png

[4] ——. Hyperbolic tangent. [Online]. Available: https://www.neuraldesigner.com/images/hyperbolic_tangent.png

[5] ——. Logistic activation. [Online]. Available: https://www.neuraldesigner.com/images/logistic.png

[6] ——. rectified linear activation. [Online]. Available: https://www.neuraldesigner.com/images/rectified-linear-activation.png

[7] ——. Performace function. [Online]. Available: https://www.neuraldesigner.com/images/performance_function.svg

[8] S. Bhatt. Generic reinforcement learning loop. [Online]. Available: https://miro.medium.com/max/700/1*7cuAqjQ97x1H_sBIeAVVZg.png

[9] Atari. Atari breakout (2600) screenshot. [Online]. Available: https://www.mobygames.com/images/shots/l/617748-atari-80-classic-games-in-one-xbox-screenshot-breakout-2600.jpg

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Clip parameter ilustration. [Online]. Available: https://arxiv.org/pdf/1707.06347.pdf

[11] G. Looye, "Basics of aeroelasticity and flutter analysis," *German Aerospace Research Center (DLR)*, 1998.

[12] L. Wang, Y. Dai, and C. Yang, "Bifurcation analysis with aerodynamic-structure uncertainties by the nonintrusive pce method," *International Journal of Aerospace Engineering*, vol. 2017, pp. 1–17, 02 2017.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[14] A. Torregrosa, L. García-Cuevas, P. Quintero, and A. Cremades, "On the application of artificial neural network for the development of a nonlinear aeroelastic model," *Aerospace Science and Technology*, vol. 115, p. 106845, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1270963821003552

[15] Siemens Digital Industries Software, "Simcenter STAR-CCM+, version 2022.1.1," Siemens 2022. [Online]. Available: https://www.plm.automation.siemens.com/global/en/products/simcenter/STAR-CCM.html

[16] Python Core Team, *Python: A dynamic, open source programming language*, Python Software Foundation, 2022, python version 3.9.16. [Online]. Available: https://www.python.org/

[17] L. Hatledal, F. Collonval, and H. Zhang, "Enabling python driven co-simulation models with pythonfmu," 2020. [Online]. Available: https://hdl.handle.net/11250/2676861

[18] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J. V. Peetz, S. Wolf, and C. Clauß, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the 8th*

International Modelica Conference, 2011, pp. 105–114. [Online]. Available: http://dx.doi.org/10.3384/ecp11063105

[19] L. Navarro García, "An artificial intelligence framework for air navigation based on deep reinforcement learning," Ph.D. dissertation, Universitat Politècnica de València, 2021.

[20] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. October, pp. 433–60, 1950.

[21] S. J. Russel and P. Norvig, *Artificial intelligence: a modern approach.* Upper Saddle River, New Jersey: Prentice Hall, 2010.

[22] What is Artificial Intelligence? How Does AI Work? [Online]. Available: https://builtin.com/artificial-intelligence

[23] T. W. Malone, D. Rus, and R. Laubacher, "Artificial Intelligence and The Future of Work," December 2020. [Online]. Available: https://workofthefuture.mit.edu/wp-content/uploads/2020/12/2020-Research-Brief-Malone-Rus-Laubacher2.pdf

[24] S. Brown. Machine Learning, explained. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[25] OpenNN. Neural network tutorial. [Online]. Available: https://www.neuraldesigner.com/learning/tutorials/neural-network

[26] ——. Training strategy tutorial. [Online]. Available: https://www.neuraldesigner.com/learning/tutorials/training-strategy

[27] S. Bhatt, "Reinforcement learning 101," Mar. 19, 2018. [Online]. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

[28] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Complete Draft).* Cambridge, Massachusetts London, England: The MIT Press, 2017.

[29] D. van der Hoff, "A Multi-Agent Reinforcement Learning Approach to Air Traffic Control," June 2020. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid:4d02e51a-0187-404d-b465-7ae01feba8e8?collection=education

[30] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[32] S. GmbH. What is cfd | computational fluid dynamics? [Online]. Available: https://www.simscale.com/docs/simwiki/cfd-computational-fluid-dynamics/what-is-cfd-computational-fluid-dynamics/#:~:text=Computational%20Fluid%20Dynamics%20(CFD)%20is,governing%20equations%20using%20computational%20power.

[33] L. Navarro García, "Numerical analysis of active aeroelastic control methods," https://github.com/LuisNavarroGarcia/placa_plana.git, 2023.

[34] A. Inc. Ray docuementation. [Online]. Available: https://docs.ray.io/

[35] ——. Understanding the ray ecosystem and community. [Online]. Available: https://www.anyscale.com/blog/understanding-the-ray-ecosystem-and-community

[36] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[37] S. D. I. Software, "Simcenter STAR-CCM+ User Guide, version 2022.1." Siemens, 2022.

[38] M. Grinberg, *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[39] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[40] "Anaconda software distribution," 2020. [Online]. Available: https://docs.anaconda.com/

[41] S. Chacon and B. Straub, *Pro git*. Apress, 2014.

[42] ——, *Pro git*. Apress, 2014.

[43] P. Quintero, *Apuntes aerodinámica*. UPV, 2020.

[44] Wikipedia contributors, "Ziegler–nichols method — Wikipedia, the free encyclopedia," 2023, [Online; accessed 8-September-2023]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Ziegler%E2%80%93Nichols_method&oldid=1140258750

[45] A. Gil, A. Tiseira, P. Quintero, and A. Cremades, "Prediction of the non-linear aeroelastic behavior of a cantilever flat plate and equivalent 2d model," *Aerospace Science and Technology*, vol. 113, p. 106685, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1270963821001954