Master Science de la Matière
École Normale Supérieure de Lyon
Université de Lyon

Stage 18/01/2024 - 18/07/2024
Corentin PRADOS
M2 Physique, Concepts and Applications

Deep Reinforcement Learning for Flow Control : application of PPO with MARL to improve drag reduction in open-channel

Abstract :
Draft : This study presents an innovative application of Deep Reinforcement Learning (DRL) to fluid mechanics, specifically focusing on drag reduction in open channel flows. The objective is to identify and mitigate Q-events, which are significant coherent structures contributing to overall drag, through strategic manipulation of the flow at the wall. By employing Proximal Policy Optimization (PPO) with Multi-Agent Reinforcement Learning (MARL), we develop a DRL framework capable of dynamically adjusting the flow to minimize drag efficiently. Our approach is integrated with ALYA, a Computational Fluid Dynamics (CFD) code that utilizes the finite element method to simulate the complex fluid dynamics accurately. The results of this study are groundbreaking, demonstrating that the DRL agent can discover more effective drag reduction strategies than those currently established in fluid mechanics literature. This breakthrough not only showcases the potential of DRL in enhancing fluid mechanics applications but also opens avenues for further research into intelligent flow control methods, promising substantial improvements in energy efficiency and system performance across a variety of engineering domains.

Key words : *Deep Reinforcement Learning (DRL), Proximal Policy Optimization (PPO), Multi-Agent Reinforcement Learning (MARL), Drag Reduction, Open Channel Flow, Q-events, Flow Control, Finite Element Method (FEM), Energy Efficiency, Environmental Engineering*

Internship supervised by :
Ricardo Vinuesa
rvinuesa@mech.kth.se

KTH Royal Institute of Technology
SE-100 44 Stockholm, Sweden

FLOW Lab
https://www.flow.kth.se

VINUESA Lab
https://www.vinuesalab.com

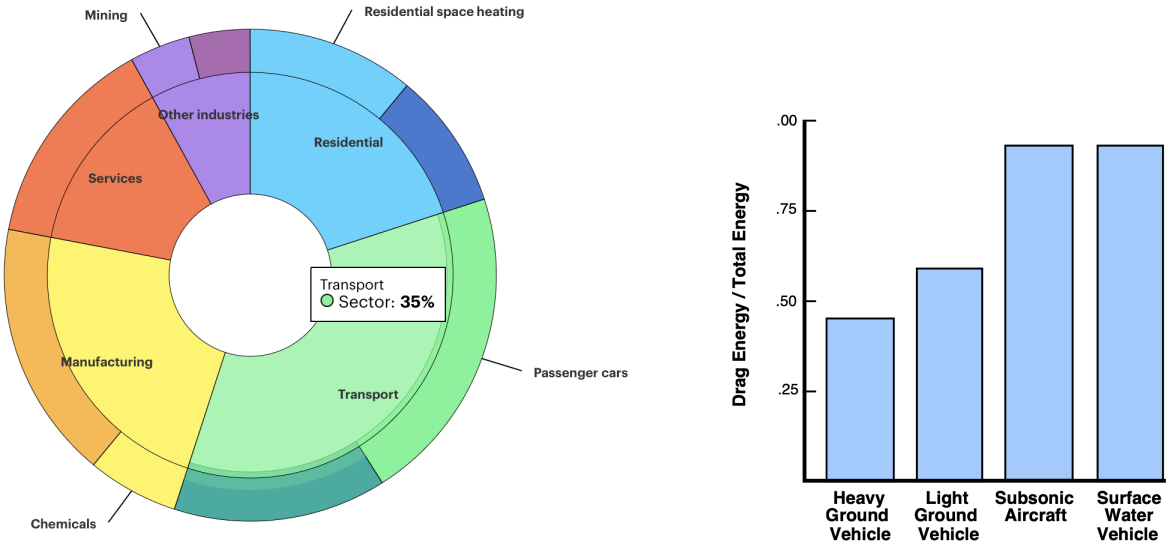
Acknowledgement

I would like to thank Ricardo for offering me this internship at KTH, which allowed me to improve my English fluency and learn extensively about turbulence, CFD, and machine learning. I am also deeply grateful to Pol and Andres for their friendly supervision throughout the internship. Additionally, I wish to thank everyone on the eighth floor for welcoming me with warmth and kindness, where I met some truly wonderful people. I also want to express my appreciation to Andrew Ng, whose excellent teaching has greatly assisted me during this internship, even though we have never met. Lastly, a big thank you to all the friends I made in Sweden who made these six months of work truly enjoyable.

Contents

Introduction

Energy consumption in the transportation sector is a critical issue, accounting for around 35% of global energy usage, as shown in FIGURE 1.4A. Addressing this problem is essential to moderate climate change and its effects [1]. A significant portion of this energy consumption is due to aerodynamic drag, which can account for nearly all the energy usage in subsonic aircraft and surface water vehicles, illustrated in FIGURE 1.4B. Consequently, reducing drag is a crucial area of research [2].



(A) (B)

Figure 1: (A) *Largest end uses of energy by sector in selected IEA countries, 2019. The transport sector is the most consuming one [1].* (B) *Energy consumed to overcome drag for ground, air, and water vehicles. Drag can account for nearly all the energy usage [2].*

Extensive research has focused on optimizing vehicle shapes to combat drag. These designs are often inspired by nature, such as the streamlined forms of sharks, which have led to innovative drag-reducing surfaces [3]. However, while shape optimizations are beneficial, they do not address the root cause of drag: turbulence.

Flow control is a promising field that seeks to understand and manage fluid flows, in order to reduce turbulence and drag for instance. This can be achieved through passive or active methods. Passive control does not require additional energy, whereas active control does [4]. A significant challenge with drag is that the faster a vehicle travels, the stronger the drag force becomes. For instance, at speeds of 50 km/h, aerodynamic drag can account for 50% of total resistance, increasing to 80% at 130 km/h. Although passive methods are useful, they are limited in their ability to influence a broad range of flow dynamics [5]. Active flow control, however, shows greater potential by allowing direct manipulation of the turbulent structures causing drag. Current active control techniques can achieve around a 25% reduction in drag by using flow ejection and suction systems to alter the near-wall flow [6]. A notable project in this domain is NASA's active flow control (AFC) mechanism, conducted on the Boeing ecoDemonstrator, a specially modified Boeing 757 aircraft. This experiment involved installing small devices on the aircraft's vertical tail that blow controlled jets of air to improve airflow.

In recent years, machine learning (ML) has opened new routes in flow control. ML can autonomously develop strategies that may not be intuitive to researchers, making it a valuable tool for advancing active flow control techniques. Scientists have applied various ML methods, including genetic programming and Bayesian optimization, to improve flow control with notable success [7, 8]. Deep reinforcement learning (DRL), a subset of ML, is particularly effective for control. For instance, DRL has been used to learn strategies for injecting flow around a cylinder, significantly reducing drag. As shown in FIGURE 2, the fluctuations in the streamwise velocity have been significantly diminished [9].

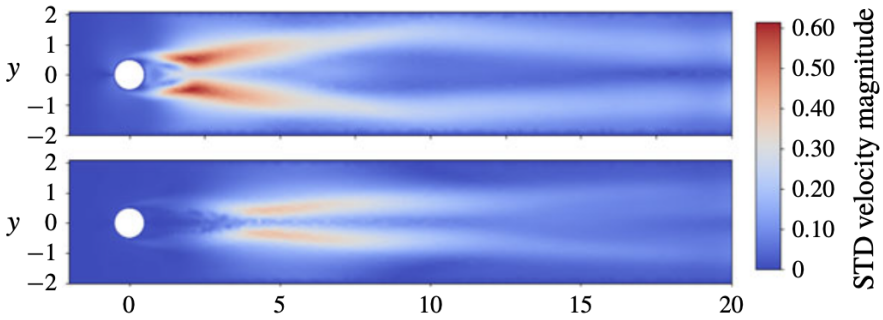


Figure 2: *Comparison of the flow morphology without (top) and with (bottom) active flow control by examining the standard deviation (STD) of the velocity magnitude using DRL in a two-dimensional simulation of the Kármán vortex street at a moderate Reynolds number (Re = 100). The artificial neural network learns an active control strategy by adjusting the mass flow rates of two jets positioned on either side of the cylinder [9].*

The project aims to apply DRL to flow control in channel flows, building on the work of Ricardo Vinuesa and Luca Guastoni. They demonstrated that DRL targeting near-wall turbulence could achieve drag reductions of 43% and 30% in minimal and larger channels, respectively, at a friction Reynolds number of 180. These results outperformed traditional opposition control methods by approximately 20 and 10 percentage points, respectively [10].

The strategy of the following project focuses not solely on near-wall turbulence and drag, but addresses the entire flow field. Specific coherent structures known as Q-events are targeted. They are crucial in turbulent flows and help to identify regions with significant velocity fluctuation. The detection criterion identifies Q-events where the product of fluctuating velocities exceeds a threshold value H as follows:

$$|u(x, y, z, t) \cdot v(x, y, z, t)| \geq H \cdot u'(y) \cdot v'(y).$$

The velocity fields used in this context include the velocity fluctuation field $u(x, y, z, t)$, representing local deviations from the mean, and the RMS velocity fluctuations $u'(y)$ of this fluctuation field. By mitigating these Q-events, the overall drag can be reduced and potentially transform turbulent flow into laminar flow, achieving a more comprehensive and effective flow control strategy [11].

In summary, the contribution of this project lies in utilizing DRL to identify and control Q-events, thereby reducing turbulence and drag in channel flows. This approach aims to make significant strides toward the ultimate goal of transforming turbulent flow into laminar flow, potentially leading to groundbreaking advancements in fluid mechanics and energy efficiency.

Following this introduction, the next section delves into the theoretical background, presenting the essential framework and notations, the concepts of coherent structures, especially Q-events, the fundamentals of computational fluid dynamics (CFD), and an introduction to deep reinforcement learning (DRL). Subsequently, the methodology section details the CFD simulations conducted, the process of data extraction and analysis, and the integration of DRL into the approach. The results follow, presenting the validation of the simulations, the detection of Q-events, and the outcomes derived from employing DRL. This is followed by a discussion that examines the implications of the findings. Finally, this report concludes with a summary of the key contributions and suggestions for future research directions.

1 Theory and framework

1.1 The World of Machine Learning in Fluid Dynamics

Deep Reinforcement Learning (DRL) has been used during this project to control fluid flow, aiming to reduce drag in channel flows. To understand how DRL works, it is crucial to have the fundamentals of machine learning. I personally attended the Andrew Ng Machine Learning Specialization series [12] to learn the basics during this project. To understand its impact on fluid mechanics, I mostly explored the works of Steven L. Brunton through numerous videos and papers as he often make a clear summary of the state of the art [13]. In this section key concepts are highlighted without the details.

1.1.1 Introduction to Machine Learning

Machine learning (ML) is a subset of artificial intelligence (AI) that allows computers to learn from data and make decisions or predictions without being explicitly programmed to perform specific tasks. It works by using algorithms to identify patterns within data and make inferences from these patterns. The power of machine learning lies in its ability to handle vast amounts of data and uncover complex patterns that are often difficult for humans to discern. Applications of machine learning span across various domains, including image and speech recognition, natural language processing, and predictive analytics [14].

There are three main types of machine learning: supervised, unsupervised, and reinforcement learning. In supervised learning, models are trained on labeled data to predict outputs from inputs. Unsupervised learning deals with unlabeled data, finding hidden patterns or structures. Reinforcement learning trains an agent to make decisions by rewarding desired behaviors and punishing undesired ones. These categories can be divided and interconnected, as illustrated in Figure 1.1.

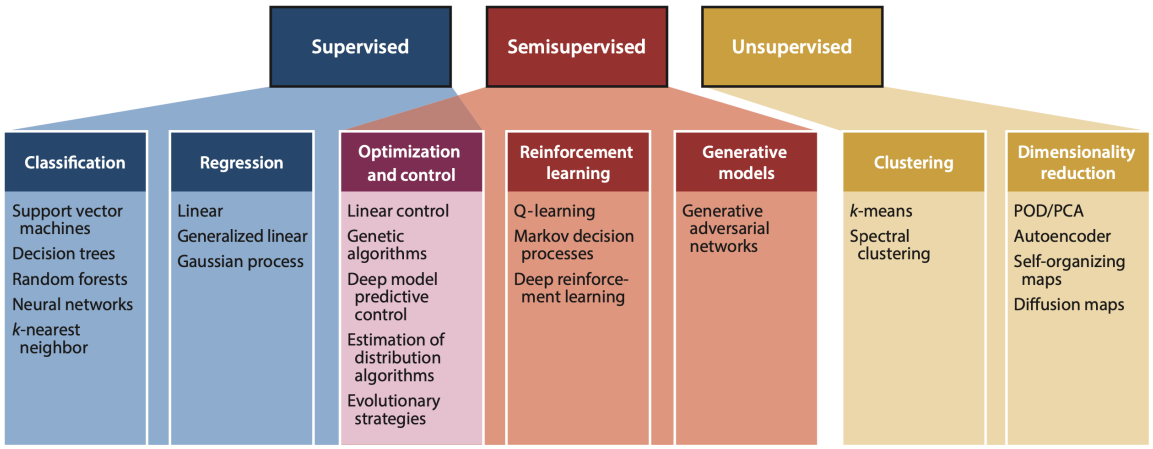


Figure 1.1: Review of machine learning categories and algorithms. Abbreviations: PCA, principal component analysis; POD, proper orthogonal decomposition [15].

1.1.2 Fundamentals of Deep Learning

Deep learning (DL) is a subset of machine learning that uses neural networks (computational architecture see next paragraph) with many layers to model complex patterns in data. Unlike classical machine learning, which often requires manual feature engineering, deep learning automatically discovers the representations needed for feature detection. This ability to automatically extract features from raw data makes deep learning particularly powerful for tasks involving large and complex datasets.

Neural networks (NN), the backbone of deep learning, are composed of layers of nodes (neurons), where each layer transforms the input data into more abstract representations. These layers include input layers, hidden layers, and output layers. Activation functions, such as ReLU or sigmoid, introduce non-linearity into the network, allowing it to learn more complex functions. The method used to update the weights of the network to minimize error is known as backpropagation [16].

1.1.3 Supervised and Unsupervised Machine Learning in Fluid Dynamics

Fluid mechanics is a high-dimensional, nonlinear, non-convex, and multi-scale problem. Luckily, patterns can emerge from this complex data. And this is exactly what machine learning is good at: extracting patterns from a huge amount of data. Unsupervised learning, in particular, can help in this matter. The classical orthogonal decomposition (POD) or principal component analysis (PCA) algorithms belong to this category of learning and are already powerful in extracting patterns from a flow. However, when these methods are combined with the power of neural networks (NN), their effectiveness is even greater. Their strength lies in their ability to implement non-linearity. For instance, Murata and his colleagues showed that a deep autoencoder with nonlinear activation functions exhibits lower reconstruction errors than POD [17]. A new method called robust PCA uses this approach to denoise a flow signal [18].

Another important property in the case of numerical and experimental fluid mechanics is the generation of a huge amount of data. This data causes significant problems in terms of energy consumption and materials used to build the servers. The development of super-resolution techniques can help with that issue. Data can be stored in low resolution, and then a super-resolution algorithm can interpolate to reconstruct the high-resolution image [19]. These kinds of machine learning algorithms can also improve data from limited sensor measurements or limited resolution CFD. Significant work has been done in this area. For example, Erichson and his team [20] created a "shallow decoder" as illustrated in Figure 1.2. It is important to be careful about the fact that this is interpolation. When extrapolation is needed, that is to say, predicting what will happen in the future, these algorithms are currently not able to do it [15]. Furthermore, generalizing these results to complex data, such as in geophysics, requires a huge amount of training data, which is currently unavailable [21].

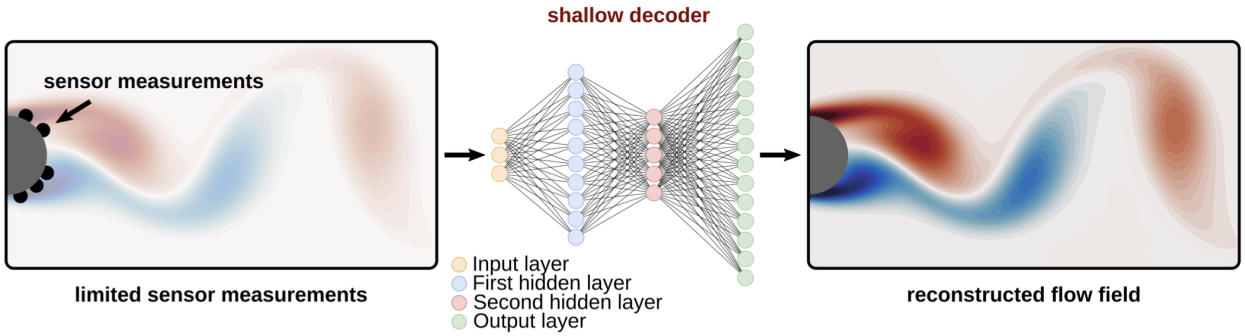


Figure 1.2: The "shallow decoder" uses a few sensor measurements to predict a detailed flow field. Starting with limited input data (input layer, here 5 sensors), the neural network processes these measurements through intermediate layers (hidden layers) to produce a comprehensive estimate of the fluid flow (output layer *ie* the reconstructed flow field). [20].

Direct Numerical Simulation (DNS), Turbulence Modeling such as Large Eddy Simulation (LES) and Reynolds-Averaged Navier–Stokes (RANS), and Reduced-Order Models (ROMs) are the three main CFD fields improved by ML. Globally, it can accelerate these simulations, improve their scaling, and enhance the physical understanding in terms of scalability and generalizability of these models [22].

In DNS, machine learning techniques, like deep neural networks, can make simulations more efficient by creating new discretization methods that better estimate spatial derivatives on low-resolution grids [23]. When it comes to turbulence modeling, methods based on machine learning, such as convolutional neural networks (CNNs), can boost the predictive power of LES and RANS models by offering more accurate subgrid-scale models for turbulent flows [24]. ROMs also gain from machine learning, as deep learning helps uncover low-dimensional manifolds that capture the core dynamics of complex flows, striking a good balance between computational efficiency and accuracy [25].

1.1.4 Reinforcement Learning Foundations

Biological systems excel at interacting with fluid environments, as seen in animals like fish, birds, and insects. For example, an eagle landing on a rock against turbulent wind can do so effortlessly, while humans struggle to understand turbulent flow. Despite the fact that the eagle only have his senses, it can navigate through complex fluid dynamics efficiently. This suggests that the eagle has learned to interact with its environment. The question is how and one of the possible answer could be through a trial and error process. This learning process is analogous to reinforcement learning (RL), which is inspired by such biological systems. It can also be compared to a dog learning a trick: if the dog performs correctly, it receives a reward, if not it does not.

More precisely RL involves training an agent to know how to interact with his environment. The agent is the learner or decision maker. The environment is everything he interacts with. He receives feedback in the form of rewards r or penalties based on its actions a at a certain state s and learns to maximize cumulative rewards. An action is what the agent can do and a state represents the environment at a specific time. The stratregy that the agent employs to determine actions according to a state is called policy, it is often written $\pi(s, a)$. Deep reinforcement learning (DRL) combines RL with DL, utilizing neural networks to approximate its policy, the parameters of the NN is written θ [26]. All of these concepts are sum up in the Figure 1.3 and will be reused in the rest of the report.

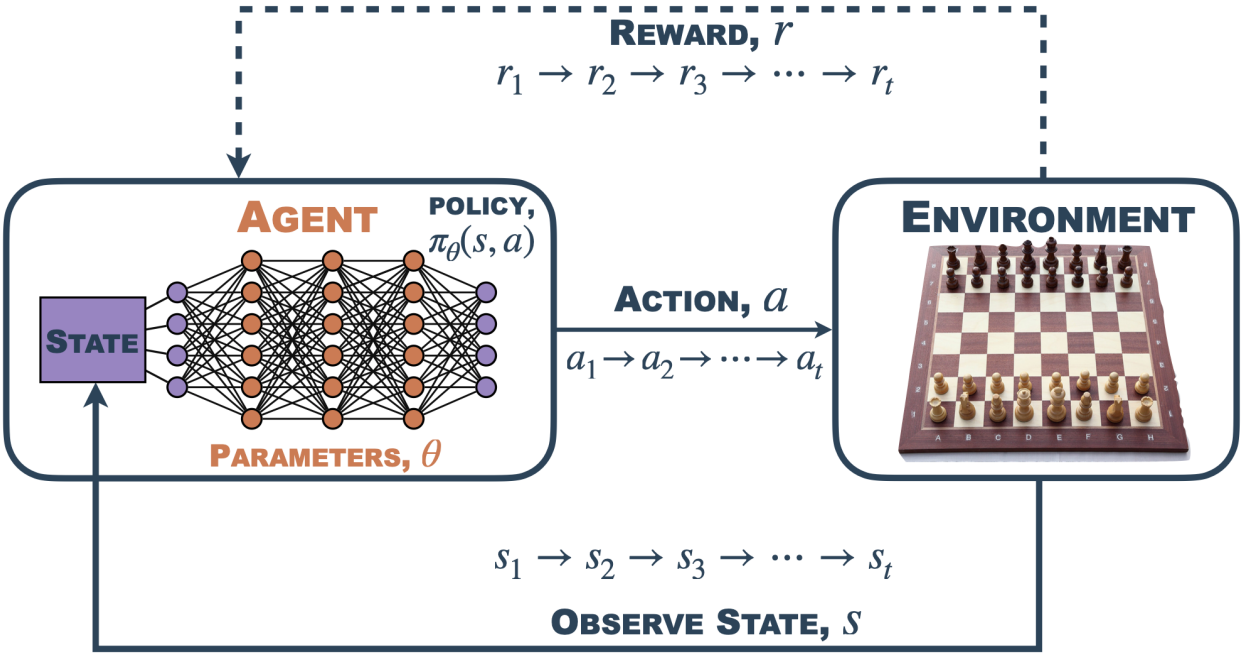


Figure 1.3: Schematic of reinforcement learning, where an agent interacts with a environment by measuring a state s and takes actions a according to a policy π that is optimized through learning to maximize future rewards r. In this case, a deep neural network with parameters θ is used to represent the policy. This is known as a deep policy network [14].

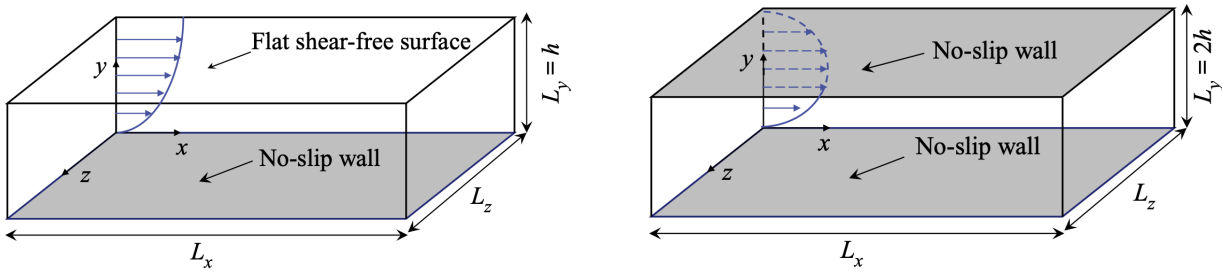
1.1.5 Applications of Reinforcement Learning in Fluid Dynamics

While reinforcement learning (RL) in fluid dynamics is still emerging compared to supervised and unsupervised learning, it shows promise in key areas. One area is modeling turbulence, where RL has been used to estimate subgrid-scale physics in large-eddy simulations (LES) with good generalization across grid sizes and flow conditions [27]. Another area is flow control, demonstrated by the application of deep RL for discovering novel strategies on aerodynamic surfaces [22], including methods for drag reduction [9], as described in the Introduction. Additionally, RL has been used to model efficient collective swimming behaviors in aquatic animals, providing insights into natural energy-efficient swimming patterns [28]. These advancements suggest that RL will play a crucial role in the future of computational fluid dynamics.

1.2 Theoretical Framework and Notations

1.2.1 Flow Configurations

Two cases will be studied: an open channel flow (OCF) and a closed channel flow (CCF), see Figure 1.4.



(A) (B)

Figure 1.4: The streamwise direction is x with length L_x, the wall-normal direction is y with length $L_y = 2h$, and the spanwise direction is z with length L_z. (A) The open channel flow (OCF) has one fixed plane at $y = 0$ and a free surface at $y = h$. (B) The closed channel flow (CCF) features two fixed plates at $y = 0$ and $y = 2h$. [29]

1.2.2 Hypotheses and Governing Equations

Under the hypotheses of mass conservation, Stokes hypothesis, incompressibility, and absence of volumic forces, the classical formulation of the Navier-Stokes equation for the flow is obtained as follows:

$$\nabla \cdot \underline{U} = 0 \tag{1.1a}$$

$$\rho(\partial_t \underline{U} + \underline{U} \cdot \nabla \underline{U}) = -\nabla P + \nu \Delta \underline{U} \tag{1.1b}$$

The variable $\underline{U} = (U, V, W)$ represents the velocity field, ρ denotes the density of the fluid, t indicates time, P refers to the pressure field, and ν stands for the kinematic viscosity.

1.2.3 Boundary conditions

Periodic boundary conditions are applied in the spanwise and streamwise directions:

$$U(x = 0) = U(x = L_x), \quad V(x = 0) = V(x = L_x), \quad W(x = 0) = W(x = L_x), \tag{1.2a}$$

$$U(z = 0) = U(z = L_z), \quad V(z = 0) = V(z = L_z), \quad W(z = 0) = W(z = L_z). \tag{1.2b}$$

For the CCF, no-slip conditions are enforced at the bottom and top walls:

$$U(y = 0) = 0, \quad V(y = 0) = 0, \quad W(y = 0) = 0, \tag{1.3a}$$

$$U(y = 2h) = 0, \quad V(y = 2h) = 0, \quad W(y = 2h) = 0. \tag{1.3b}$$

For the OCF, no-slip conditions are applied at the bottom wall, and symmetry boundaries at the top:

$$U(y = 0) = 0, \quad V(y = 0) = 0, \quad W(y = 0) = 0, \tag{1.4a}$$

$$V(y = h) = 0, \quad \frac{\partial U}{\partial y}(y = h) = 0, \quad \frac{\partial W}{\partial y}(y = h) = 0. \tag{1.4b}$$

1.2.4 Non-Dimensionalization

The validation of the CFD code is pursued through a comparison with the results presented by Jiménez and Moin [30]. For consistency, the same normalization method described in their study will be employed.

Notation

The notation y_{wall} will be used for $y = 0$ in the case of OCF and for $y = 0$ and $y = 2h$ in the case of CCF. In practice, in the latter case, a value $X(y_{\text{wall}})$ would be equal to $\frac{X(0)+X(2h)}{2}$, *ie* the mean between the two walls.

Characteristic Velocity

The local friction velocity, u_τ, is a critical parameter in characterizing near-wall turbulence. It is a measure of the shear stress exerted by the fluid flow on the wall and is defined at the wall as:

$$u_\tau(y_{\text{wall}}) = (\overline{\Omega}(y_{\text{wall}})\nu)^{1/2},$$

where $\overline{\Omega}$ represents the time-averaged wall shear stress, calculated from:

$$\Omega(y_{\text{wall}}) = \frac{1}{L_x L_z} \int_0^{L_x} \int_0^{L_z} \left.\frac{\partial U}{\partial y}\right|_{(x,y_{\text{wall}},z)} dx\, dz.$$

Characteristic Length and Time

A characteristic length scale, δ_τ, also known as the viscous length scale or wall unit, can be defined for turbulent flows. Additionally, a local 'wall' time, t_τ, represents the minimum timescale for events occurring in the wall layer. These scales are given by:

$$\delta_\tau = \frac{\nu}{u_\tau} \quad \text{and} \quad t_\tau = \frac{\delta_\tau}{u_\tau}.$$

The viscous length scale δ_τ represents the distance from the wall where viscous effects dominate, while the wall time t_τ is the time it takes for the turbulent eddies to traverse this distance.

Non-dimensional Length, Velocity, and Time

To facilitate comparisons, non-dimensional coordinates, velocities, and time defined in wall units are employed as they are commonly used in the literature. The non-dimensional coordinates, velocities, and time are expressed as:

$$y^+ = \frac{y}{\delta_\tau}, \quad U^+ = \frac{U}{u_\tau}, \quad \text{and} \quad t^+ = \frac{t}{t_\tau}.$$

This normalization applies to all coordinates, velocity components, and time scales, facilitating a consistent and standardized description of the flow. Utilizing these non-dimensional variables, such as y^+, U^+, and t^+, is particularly advantageous for analyzing near-wall turbulence phenomena.

Reynolds Number in Practice

In the study of turbulent flows, the Reynolds number is a fundamental dimensionless quantity that characterizes the flow regime. For wall-bounded turbulent flows, the friction Reynolds number, Re_τ, is commonly used. It is defined as:

$$Re_\tau = \frac{u_\tau h}{\nu}.$$

1.2.5 Relevant Variables

The notation $\langle\cdot\rangle_{x,z,t}$ represents the averaging operation over x, z, and t. The time-averaged velocity field $\overline{\underline{U}}$ is:

$$\overline{\underline{U}} = \langle \underline{U} \rangle_{x,z,t}\,.$$

The velocity fluctuation field, $\underline{u}(x, y, z, t)$, is then expressed as:

$$\underline{u}(x, y, z, t) = \underline{U}(x, y, z, t) - \overline{\underline{U}}(y). \tag{1.5}$$

From $\underline{u}$, the root mean square (RMS) velocity fluctuations are defined as:

$$\underline{u'}(y) = \sqrt{\langle \underline{u}^2(x, y, z, t) \rangle_{x,z,t}}. \tag{1.6}$$

1.3 Coherent Structures in Turbulent Flows

Coherent structures are fundamental elements in the study of turbulence. These structures are spatially localized, temporally persistent, and play a critical role in the transport of momentum, energy, and scalar quantities within turbulent flows. Unlike the random and chaotic nature of turbulence often described statistically, coherent structures can be considered as organized patterns that emerge naturally within the turbulent flow field. They exhibit distinct shapes and dynamics, allowing them to stand out clearly against the otherwise chaotic background of turbulence [31]. For example earth climate, ocean and atmosphere are highly turbulent. For example, Earth's climate, oceans and atmosphere, are highly turbulent, mixing and transporting materials across the globe. Despite this turbulence, it is possible to observe distinct structures within these flows, as shown in Figure 1.5.

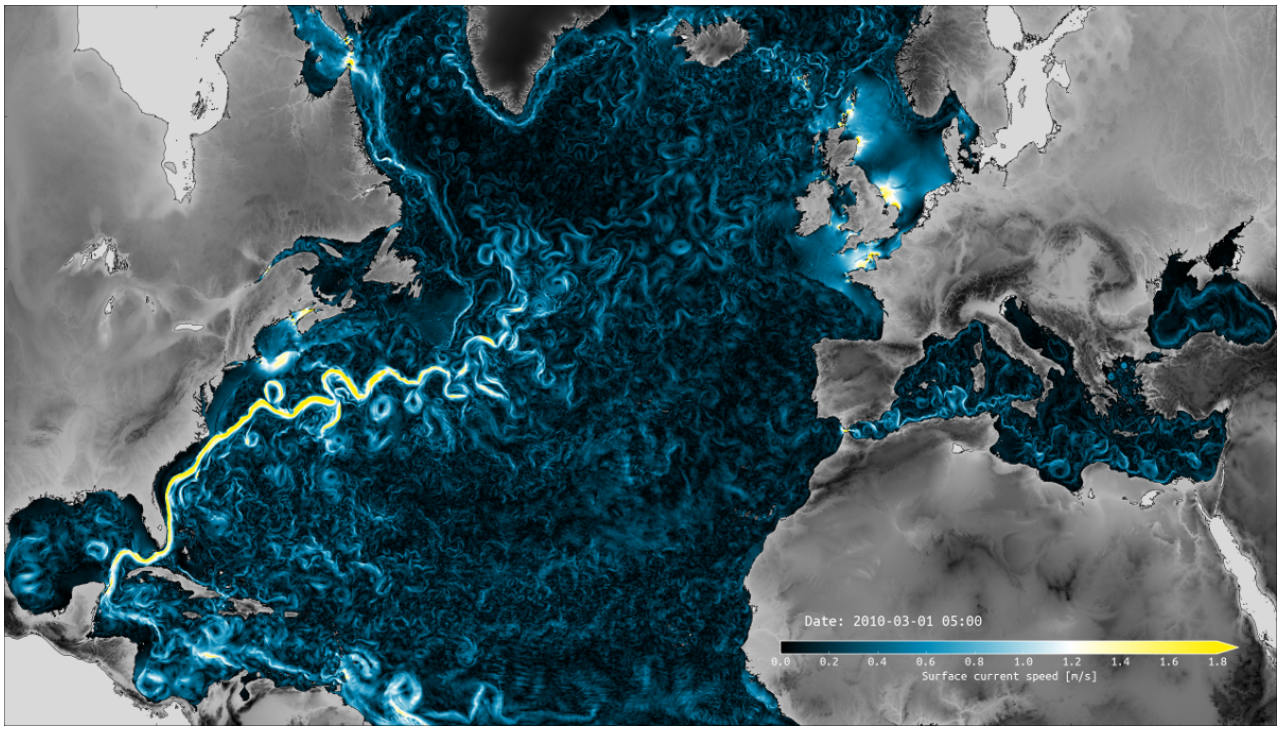


Figure 1.5: Frame of hourly surface current speed in the eNATL60 simulation of the Atlantic Ocean with explicit tidal motion. Footprints of tidally-generated internal waves are visible in the southernmost part of the domain and around the Azores. Despite the turbulent nature of the flow, patterns such as eddies remain discernible. Observations from the simulation video available on the author's website show that these eddies persist over time. [32].

The concept of coherent structures changes the view of turbulence from being purely stochastic to a more deterministic perspective. Understanding these structures provides a clearer insight into turbulent dynamics. These structures are not random; they follow specific patterns and rules dictated by fluid dynamics equations. This organized behavior offers an opportunity to simplify the complex nature of turbulence by focusing on these coherent patterns. In this work, these structures will serve as the foundation for the agent's reward system.

1.3.1 Coherent Structures in Channel Flows

Closed Channel Flow Dynamics

In closed channel flows (CCF), coherent structures are typically identified as eddies with sufficient internal dynamics to function independently of the surrounding turbulent flow. These structures are crucial for understanding channel flow turbulence as a high-dimensional dynamical system, characterized by their ability to sustain themselves over time and significantly influence the flow's energy dynamics. Coherent structures in CCF include streamwise vortices, hairpin vortices, and large-scale motions. These structures play a vital role in transporting momentum and energy across the channel. Streamwise vortices, for example, contribute to

turbulence regeneration by lifting low-speed fluid away from the wall and bringing high-speed fluid towards it. They interact with each other and with the mean flow, leading to complex dynamics essential for understanding turbulence in these settings [31].

For a comprehensive understanding of coherent structures in channel flows, including visual representations, refer to studies and visualizations by Adrian [33].

Open Channel Flow Dynamics

In open channel flows (CCF), which are characterized by a free surface, exhibit coherent structures that are influenced by both the wall and the free surface. The dynamics in open channel flows are further complicated by the presence of surface waves and the interaction between the air and water phases.

In open channel flows, coherent structures such as surface rollers, streaks, and large-scale vortices play significant roles. Surface rollers are formed near the free surface due to the interaction between the flow and surface waves, while streaks and vortices are similar to those observed in CCF but are modulated by the presence of the free surface [34].

Comparative Analysis of CCF and OCF Structures

There is a strong link between coherent structures in CCF and OCF. While the fundamental mechanisms driving the formation of these structures are similar, the presence of a free surface in OCF adds a layer of complexity and brakes a symmetry. Surface tension, wave interactions, and the exchange of momentum between the upper and down phases all influence the behavior of coherent structures in OCF. This interaction can alter the size, shape, and dynamics of these structures compared to those found in CCF, leading to unique patterns of turbulence that are important for applications such as river engineering and environmental fluid mechanics [34]. In this study, the top layer of the OCF is fixed using a symmetry condition, minimizing the effects of the free surface. This approach makes the behavior of the OCF closely resemble that of a mid-channel CCF.

Identifying Classical Coherent Structures

Most commonly used criteria are the Q criterion [35], based on the second invariant of the velocity-gradient tensor; the Δ criterion [36], based on the second and third invariants of the velocity-gradient tensor; and the $\lambda 2$ criterion [37], which derives new expressions from the gradient of the Navier-Stokes equations. These criteria are classical methods for identifying vortical structures, as demonstrated by del Álamo et al. [38], who showed that vortex clusters can be either detached or attached to a wall.

A more recent criterion for identifying coherent structures is based on a strong correlation between u and v fluctuations, known as Q-events, which will be utilized in this work.

1.3.2 Q-Events in Turbulent Flows

Definition and Importance

Q-events refer to specific areas within turbulent flows that are significantly involved in momentum transfer and the production of turbulent kinetic energy (TKE). They are particular types of coherent structures. These regions are essential for understanding overall drag within the system, making them a critical area of study. Q-events are identified as coherent zones with high instantaneous Reynolds stress, defined by the condition:

$$u(x, y, z, t)v(x, y, z, t) > Hu'(y)v'(y),$$

where H is a threshold function and the velocities are as defined in the previous section. This criterion highlights areas with notable Reynolds stress [39].

Quadrant Analysis Approach

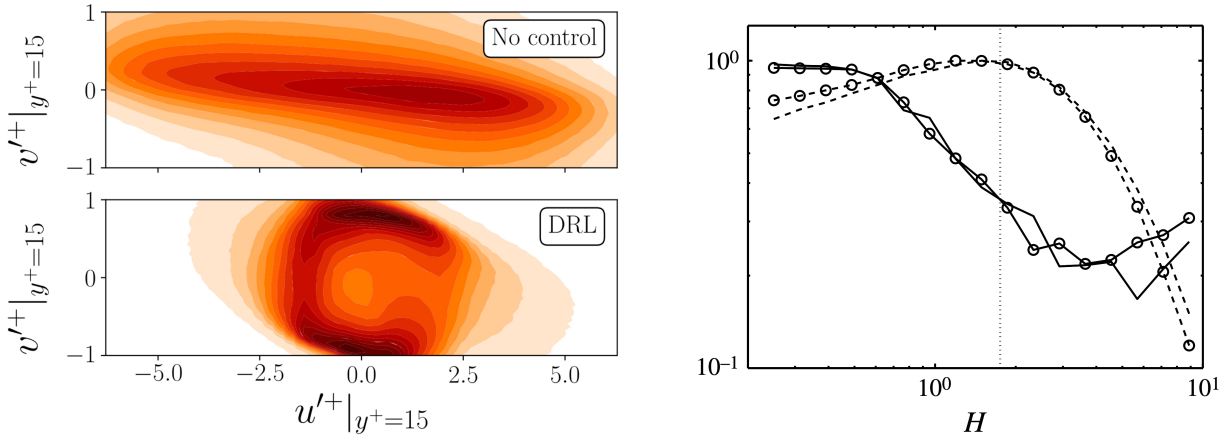
Quadrant analysis is a method used to examine Q-events by categorizing contributions to Reynolds shear stresses, initially introduced by Wallace et al. [40]. It involves plotting the (v', u') fluctuations to divide streamwise and wall-normal fluctuations into four categories: Outward Interactions (Q1), where $u' > 0$ and $v' > 0$; Ejections (Q2), featuring $u' < 0$ and $v' > 0$; Inward Interactions (Q3), with $u' < 0$ and $v' < 0$; and Sweeps (Q4), characterized by $u' > 0$ and $v' < 0$ [41]. Three dimensional extension of the classical quadrant analysis is possible. It is important to note that in a turbulent flow ejection and sweeps are the dominant events.

This analysis effectively reveals the relationship between fluctuation distributions and turbulent flow characteristics. For example, in Guastoni's paper, it demonstrates changes in inner-scaled velocity-fluctuation distributions in OCF with and without DRL control, as shown in Figure 1.6A [10].

Percolation Analysis for Optimizing Q-Event Detection

Percolation analysis is used to optimize the detection of Q-events by plotting their detection rate against the parameter H. This analysis aims to find the H value that maximizes the rate of detection. Using an identification method with a constant threshold can be problematic in inhomogeneous flows. Instead, H is chosen based on its percolation behavior, ensuring the capture of the most significant Q-events [42, 43].

Percolation theory, which examines the statistics of connected components in a random graph, is applied here to understand how the volume of connected objects varies with H. This method helps identify the optimal H that balances the detection of both small, strong events and larger, more distributed ones. The percolation threshold, where the volume of the largest identified object rapidly increases while the number of objects decreases, typically occurs within the range $0.8 \leq H \leq 3$, regardless of the Reynolds number. Although this result has not been thoroughly verified in the literature due to the unavailability of relevant papers, it is illustrated in Figure 1.6B, based on the work of Lozano-Duran and Jimenez [44]. The percolation diagram shows minimal variation between Reynolds numbers $Re_\tau = 934$ [45] and $Re_\tau = 2003$ [46], despite the significant difference in their values. This consistency suggests that the observed percolation behavior is robust across different Reynolds numbers.



(A) (B)

Figure 1.6: (A) Distribution of the inner-scaled velocity-fluctuation components after the initial transient for the uncontrolled case (top) and when using DRL (bottom). The distribution of the events changes completely after the action of the DRL agent [10]. (B) Percolation diagram for the identification of Q-events. ——, Ratio of the volume of the largest object to the volume of all identified objects; – – –, ratio of the number of identified objects to the maximum number of objects. To maximize the second ratio, an H around 1 seems optimal [44]. Lines without symbols correspond to $Re_\tau = 934$ [45], and those with symbols correspond to $Re_\tau = 2003$ [46].

Application of Q-events in the Literature

Lozano-Durán et al. [47] demonstrated in 2012 that various structures, such as attached vortex clusters, attached Q2 events, and attached large-scale Q2 events, can be detected in three-dimensional turbulent flows. These structures are closely related to turbulence dynamics. Atzori et al. applied the study of Q-events to more complex flows, first in 2021 in a duct and then in 2022 in a turbulent wing, showing their relevance even in highly complex turbulent environments [34, 48].

These demonstrations of effectiveness in the literature provide confidence in the efficiency of targeting Q-events to reduce overall turbulence, particularly through the application of Deep Reinforcement Learning (DRL) in the case of the following work. The ultimate goal is to determine whether reducing Q-events can lead to the laminarization of the flow, although it remains uncertain if this is theoretically possible.

2 CFD Simulations with Alya

2.1 Numerical Tools

2.1.1 The Solver: Alya

Alya is a computational fluid dynamics (CFD) software framework developed by the Barcelona Supercomputing Center (BSC). It is designed to solve complex fluid dynamics problems using high-performance computing. Alya employs the finite element method (FEM) to discretize and solve the governing equations of fluid flow, specifically the dimensional incompressible Navier-Stokes equations. This method is particularly effective for handling complex geometries and varying boundary conditions.

The choice of Alya for this project is motivated by several factors. Firstly, Alya's parallel computing capabilities make it suitable for large-scale simulations. Secondly, the efficiency of the FEM in dealing with complex geometries aligns well with the project's requirements. Furthermore, Alya has a proven track record of successful integration with deep reinforcement learning (DRL), as demonstrated in previous studies by Suárez et al. [49]. Its specific design for fluid dynamics applications and ability to handle various boundary conditions, such as implementing wall jets for flow control, further justifies its selection.

2.1.2 The Mesh Generator: Gmsh

Meshing is the process of dividing a geometric domain into discrete elements used to approximate the domain for numerical simulations. For this project, a Chebyshev distribution in the y-direction and a regular distribution in the x and z-directions are used. This choice provides better resolution near the walls where gradients are steep.

The element order, referring to the polynomial degree used to approximate the variations of flow variables within each element, is set to first-order (linear) due to the simple geometry of the domain (a cuboid). Gmsh is selected as the mesh generator because it is widely used in CFD, compatible with Alya, and open-source, making it an ideal tool for this project.

2.1.3 The Viewer: Paraview

Paraview is an open-source software designed for the analysis and visualization of large-scale data from numerical simulations. It is chosen for its advanced 3D visualization capabilities, extensive post-processing tools, support for a wide range of file formats used in CFD, and its intuitive user interface. Paraview is a standard tool in the CFD community, making it well-suited for this project's needs.

2.2 Simulation Setup

2.2.1 Introduction

The primary objective of this project is to reproduce the results of Luca [10]. The key parameters used in his study are summarized in the Table 2.1.

Parameter	Explanation	Luca's Choice	Jimenez's Choice
Re	Reynolds number	180	$Re_\tau = 180$
Ly	Height of the domain	h	$2h$
Lx	Length of the domain	$2.67h$	$12\pi h$
Lz	Width of the domain	$0.8h$	$4\pi h$
Ny	Resolution in the y-direction	65	$\Delta y^+ = 6.1$ (max scale step)
Nx	Resolution in the x-direction	16	$\Delta x^+ = 7 - 8$
Nz	Resolution in the z-direction	16	$\Delta z^+ = 3 - 4$
CFL	CFL safety factor	1	0.7

Table 2.1: Summary of the parameters used in the paper by Luca *et al.* [10] and the paper by Jiménez *et al.* [50].

Since a different solver is used, it is necessary to validate the simulation before employing it in a DRL program. This validation is performed by comparing the simulation results with those of Jimenez [50], a well-known reference in the field of CFD who conducted simulations at the same Reynolds number. The parameters used in Jimenez's simulations are also listed in the table Table 2.1.

2.2.2 Simulation Configuration

The chosen domain for this project is a 3D channel flow with a Chebyshev distribution in the y-direction and regular distribution in the x and z-directions, using first-order elements due to the simple cuboid geometry. The parameters for the simulation are $Re_\tau = 180$, $Lx = 6.67h$, $Ly = 2h$, $Lz = 0.8h$, $Nx = 48$, $Ny = 192$, and $Nz = 48$. The kinematic viscosity $\nu = 3.550 \times 10^{-4}$ and density $\rho = 1$ are chosen to achieve the desired Re_τ, with no physical meaning in this context. As there is no physical meaning, physical units are not used throughout the project. The focus is on dimensionless numbers, such as the Reynolds number.

A CFL safety factor of 1 is used initially, with adjustments made as needed. The initial condition is a stationary state of the Poiseuille flow at a low Reynolds number in 3D, with a 5% perturbation on the streamwise velocity, and the flow is driven by a fixed flow rate.

To achieve better generalization and lower computational costs, studying the OCF case would have been ideal, as used by Luca. However, due to boundary condition configuration issues in Alya, the focus is on the CCF case. Notably, Jimenez's results were also obtained in the CCF case.

2.2.3 Data Processing and Analysis

Data from Alya is converted from mpio.bin format to vtk format for analysis. Visualization is performed using Paraview to interpret the vtk files. Post-processing is conducted using Python with pandas for normalization and further analysis of the data. The post-processing part is detailed and can be found on the GitHub project at the following address: `https://github.com/corentinprados/Q_event_DRL_control.git`.

2.3 Simulation Results

In the analysis, the focus is not on the transition state between the initial imposed state and the developed turbulence state. The analysis begins after this transient period. According to Jiménez *et al.*, at least 10 wash-out times are required to achieve reliable statistics. The wash-out time is calculated as $t_w = \frac{L_x}{U_b}$, where U_b is the bulk velocity, defined as the average velocity across the cross-sectional area of the channel. The start time is denoted as t_i, and the end time is denoted as t_f. The analysis is considered from t_i until time t_f.

When values are not accompanied by a + sign, they are in Alya units, *i.e.*, dimensional units. While it would be more convenient to use dimensionless units exclusively, there are instances where referring to dimensional units is more practical.

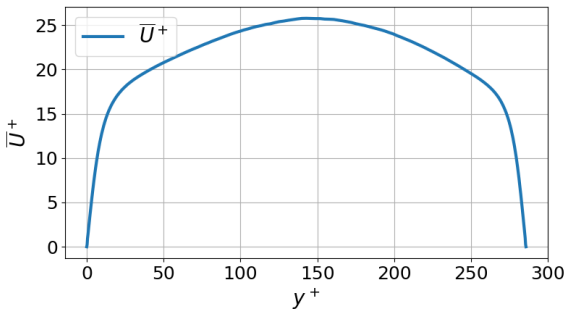
2.3.1 General Statistical Outcomes

The statistics of this simulation are derived from 274 snapshots taken at regular intervals within the timeframe from $t_i = 891.7$ to $t_f = 1376$.

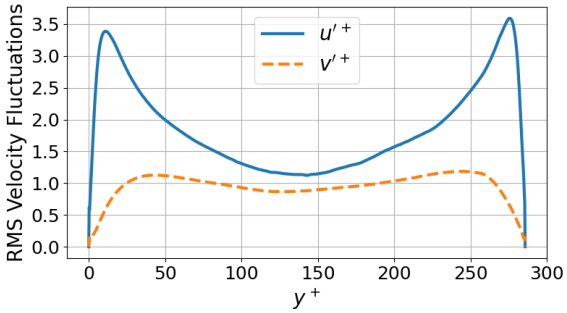
For the characteristic values, the simulation yields the following: $u_\tau = 5.068 \times 10^{-2}$, $\delta_\tau = 6.999 \times 10^{-2}$, and $\mathrm{Re}_\tau = 143$. The obtained Re_τ of 143 is lower than the desired 180, indicating a need to adjust the parameters or correct any errors in the implementation.

The wash-out time is $t_w^+ = 22$. Upon conversion, $t_i^+ = 6456$ and $t_f^+ = 9967$. Thus, the condition to achieve reliable statistics is met.

The average streamwise velocity profile and the RMS velocity fluctuation profiles for the streamwise and wall-normal speeds are shown in Figure 2.1. The average streamwise velocity profile exhibits a parabolic shape with a steeper gradient near the wall, contrasting with the fully parabolic shape of the stationary state of Poiseuille flow at a low Reynolds number. The RMS velocity fluctuation profiles show strong amplitude near the wall, diminishing further away. The amplitude of the streamwise fluctuation is higher than that of the wall-normal fluctuation. These shapes of the statistics align well with the Jiménez's ones. It is now interesting to compare the obtained values quantitatively with the literature.



(A) Mean Velocity Profile

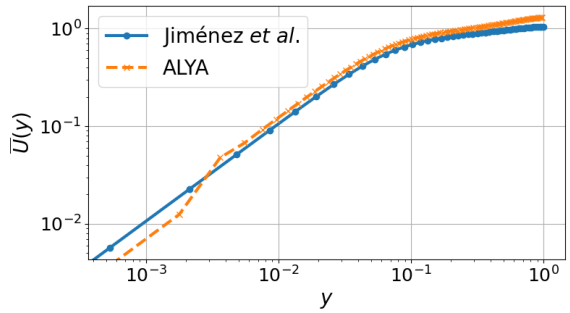


(B) RMS Velocity Fluctuation Profiles

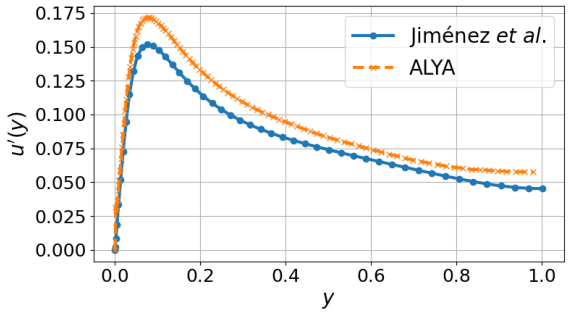
Figure 2.1: Velocity profiles for the interval $t_i = 891.7$ to $t_f = 1376$ of the Alya simulation. The average streamwise velocity profile shows a steeper gradient near the wall, while the RMS velocity fluctuation profiles have higher amplitudes near the wall and decrease further away. The streamwise fluctuation amplitude is higher than the wall-normal fluctuation. The shapes are qualitatively consistent with Jimenez's results [50].

2.3.2 Comparison with Jiménez's Data

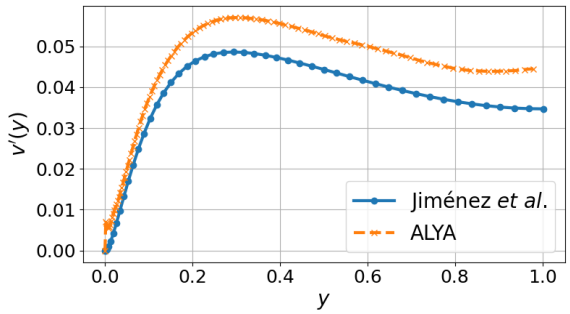
The comparison with Jimenez's statistics is presented in Figure 2.2. Since the characteristic length and velocity scales are not the same, with Jimenez's values being $u_{\tau,\text{Jiménez}} = 0.57231059 \times 10^{-1}$ and $\delta_{\tau,\text{Jiménez}}$, the decision was made to keep the dimensional values.



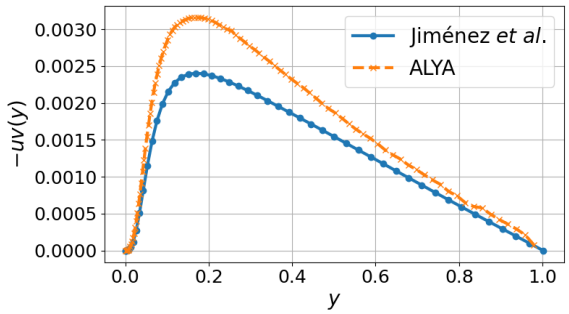
(A) Mean Streamwise Velocity Field



(B) RMS Streamwise Velocity Fluctuations



(C) RMS Wall-normal Velocity Fluctuations



(D) Reynolds Shear Stress

Figure 2.2: Comparison with Jimenez Statistics for the analysis of the simulation from $t_i = 891.7$ to $t_f = 1376$: (A) Mean Streamwise Velocity Field. (B) RMS Streamwise Velocity Fluctuations. (C) RMS Wall-normal Velocity Fluctuations. (D) Reynolds Shear Stress.

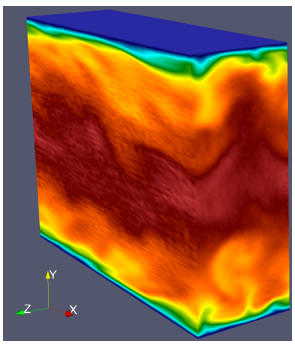
In all the figures, the overall shapes are consistent with Jimenez's results. However, a discontinuity is observed at small y in the mean streamwise velocity field in the Alya simulation, and a small discontinuity is present at small y in the RMS wall-normal velocity fluctuations. Regarding the values, a global shift results in the amplitudes from Alya being stronger than those from Jimenez. This difference is even more pronounced in the Reynolds shear stress due to the multiplicative nature of the quantities involved.

The overall shift is expected due to the lower Reynolds number used in the Alya simulation. However, for the mean streamwise velocity field, the amplitude should be smaller than Jimenez's, as it is known that the mean streamwise velocity field increases, especially the center velocity, with Reynolds number [50]. For the RMS velocity fluctuations, higher Reynolds numbers should result in higher curves away from the wall, which is not observed in the current results.

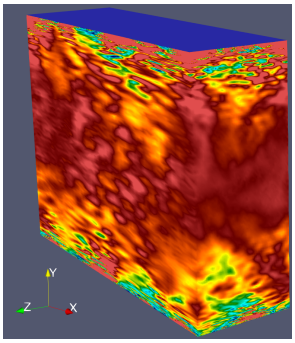
The results indicate discrepancies between the Alya simulation and Jimenez's data. While some differences are expected due to the varying Reynolds numbers, these discrepancies cannot be fully explained by this difference alone, suggesting potential issues in the simulation.

2.3.3 Assessment of Numerical Artifacts via Paraview Renderings

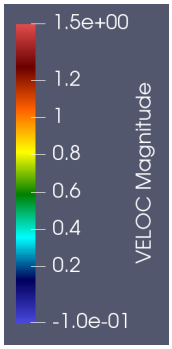
To gain a clearer understanding, the Paraview renderings were examined. Numerical artifacts seems to be observed at certain points, forming and then dissipating, as shown in FIGURE 2.3.



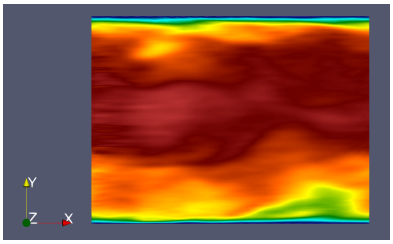
(A) 3D visualization of $U(t = 901.7)$.



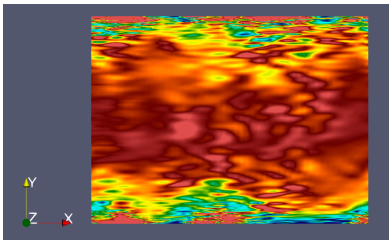
(B) 3D visualization of $U(t = 1041)$.



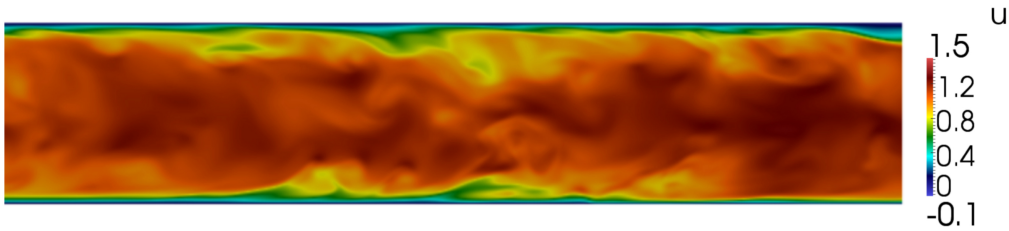
(C) Colorbar for (A), (B), (D), (E).



(D) 2D visualization of $U(t = 901.7, z = \frac{L_z}{2})$.



(E) 2D visualization of $U(t = 1041, z = \frac{L_z}{2})$.



(F) 2D visualization of U in the plane (X,Y) of a fully developed turbulence CCF at $Re_\tau = 180$ from Lee *et al.* [51].

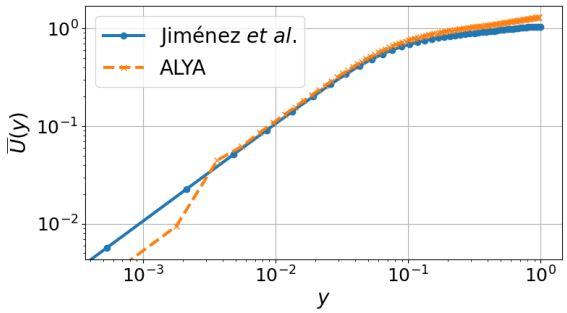
Figure 2.3: Visualization of numerical artifacts in the simulation compared with data from Lee *et al.*. The simulation at $t = 901.7$ appears accurate, while at $t = 1041$ the simulation shows signs of instability. This numerical artifact dissipates after a few timesteps.

When compared with the image provided by Lee *et al.* [51], it can be observed that at $t = 901.7$, the simulation appears correct, whereas at $t = 1041$, the simulation shows signs of artefacts. The 3D view shows only the edges of the domain; hence, a 2D view is provided, revealing that these numerical artifacts persist even in the middle of the flow. For better visibility, the subsequent time field is not shown, but it resembles the field at $t = 901.7$, indicating that the artifacts have been smoothed out. These artifacts form and dissipate throughout the simulation, leading to significant variations, which may explain the higher amplitudes in Alya's results. To verify this hypothesis, it would be useful to study the statistics over intervals without these artifacts.

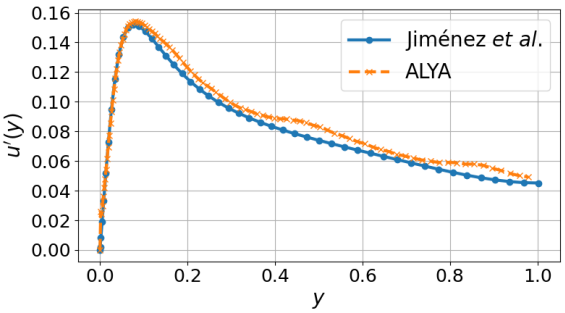
2.3.4 Analysis of Truncated Interval for Improved Statistics

A truncated interval was manually selected by examining the 3D visualization of the flow. For this new interval, the values obtained are $u_\tau = 4.438 \times 10^{-2}$, $\delta_\tau = 7.991 \times 10^{-2}$, and $\mathrm{Re}_\tau = 125$. This Re_τ is even smaller. The truncated interval is from $t_i = 895.2$, i.e., $t_i^+ = 4972$, to $t_f = 935.3$, i.e., $t_f^+ = 5195$. The new statistics are compared with those of Jimenez in Figure 2.4.

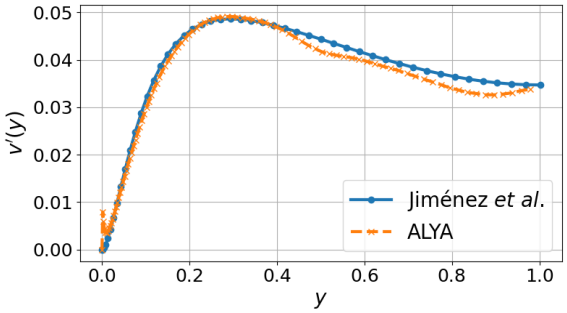
The results in terms of values are more consistent than the previous ones, even though they are still too high, for example, in the mean streamwise velocity field. Given the smaller Reynolds number, the center amplitude should decrease and be lower than that of Jimenez's results. Furthermore, there are still discontinuities at low y. Oscillations can also be observed at higher y. These may be potentially due to the fact that the number of snapshots is too low to perform reliable statistics over this interval, as their number is only 20.



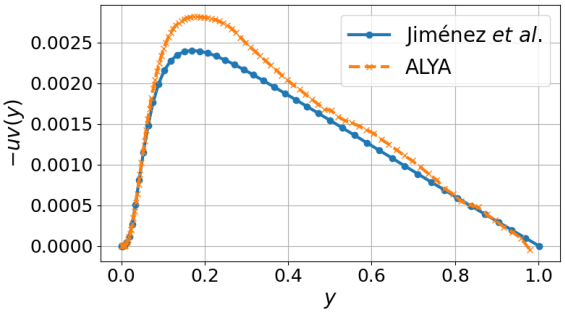
(A) Mean Streamwise Velocity Field



(B) RMS Streamwise Velocity Fluctuations



(C) RMS Wall-normal Velocity Fluctuations



(D) Reynolds Shear Stress

Figure 2.4: Comparison with Jimenez Statistics for the analysis of the simulation from $t_i = 895.2$ to $t_f = 935.3$: (A) Mean Streamwise Velocity Field. (B) RMS Streamwise Velocity Fluctuations. (C) RMS Wall-normal Velocity Fluctuations. (D) Reynolds Shear Stress.

The results in stable intervals show some improvement but are still not satisfactory. Further investigation is needed to identify the source of discrepancies, which could be due to unsuitable parameters, issues in our implementation, or problems within the Alya code itself.

2.3.5 Challenges and Limitations in Using ALYA

Alya operates as a black box, with its documentation being outdated and lacking information on many parameters. Within the team, only one member had prior experience with Alya, and even they faced difficulties with its usage. Several attempts were made to optimize the simulation on a 10-core workstation, as access to the supercomputer was unavailable due to maintenance. Efforts included trying Jimenez's resolution, which was too time-consuming, reaching only $t = 137$ over two weeks and encountering mesh initialization issues. Another attempt with a CFL safety factor of 0.2 progressed very slowly, taking one week to reach $t = 122$, and frequent crashes further delayed progress. Extending the initial simulation from $t = 900$ to $t = 1400$ took almost two weeks, with multiple restarts due to crashes, indicating persistent instability issues. Needing to continue work on Q-events and deep reinforcement learning (DRL), the Alya simulation was paused. At the end of the internship, the supervisor reassigned the work to another team member to distribute the workload more effectively.

3 Q-events analysis

The results of this section are based on the same data as the previous part, Section 2.3, namely 274 snapshots taken at regular intervals within the timeframe from $t_i = 891.7$ to $t_f = 1376$. As previously explained, these data are not optimal, and the simulation still needs improvement. However, to advance the project, it was still valuable to develop the section on Q-events analysis to better understand them and prepare for the integration with the future reinforcement learning program.

The specific time chosen for Q-events visualization is still $t = 901.7$. This will allow readers to visualize this moment of the simulation by comparing the results with Figure 2.3, facilitating a better visual understanding of what a Q-event is. Oopening the snapshot with Paraview remains more effective for seeing all the layers of the simulation and better understanding the concept.

3.1 Identification of Q-events

3.1.1 Methodology of detection

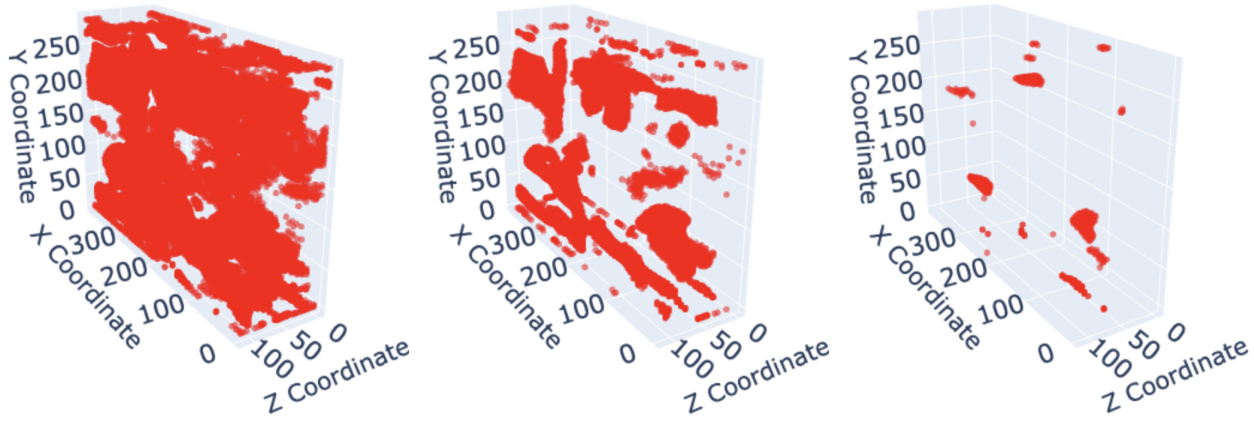
For a given time, the method is to examine each point on the grid to determine if the condition for Q-events defined in the Section 1.3 is met *ie* :

$$u(x, y, z, t)v(x, y, z, t) > Hu'(y)v'(y),$$

where H is a threshold function. Their number and volume highly depend on the threshold parameter H. A cluster of Q-events will be defined as a connected group of points that satisfy the Q-event condition.

3.1.2 Visualization of Results

In Figure 3.1, a 3D visualization is available to understand better the shape, number, and volume of the Q-events, as function of different values of H.



(A) $H = 0.8$. (B) $H = 2$. (C) $H = 5$.

Figure 3.1: Detection of Q-events at time $t = 901.7$ for different values of the parameter H. Higher values of H result in fewer detected Q-events. If H is too small, all points are classified as Q-events, resulting in a single cluster. Conversely, if H is too large, no points meet the Q-event criteria.

These results illustrate the relationship between the threshold parameter H and the Q-events. If H is too small, all points are classified as Q-events; conversely, if H is too large, no points meet the Q-event criteria. Regarding clusters, if H is too small, all points being Q-events results in a single cluster. As H increases, Q-events begin to separate, leading to an increase in the number of clusters. However, if H is excessively large, no Q-events are detected. This highlights the concept of balance, which will be further examined using the percolation diagram.

3.2 Cluster Analysis of Q-events

3.2.1 Clustering Algorithm

Once the Q-events have been identified, an algorithm is required to detect clusters and determine their number. These measurements are crucial for the DRL program's reward function, which aims to reduce either the volume or the number of clusters to ultimately decrease the overall Q-events. Directly using Q-events for the machine learning program might seem logical, but the primary source of turbulence lies in the large structures, i.e., significant clusters. Therefore, reducing clusters is more beneficial from a turbulence perspective than reducing individual Q-events.

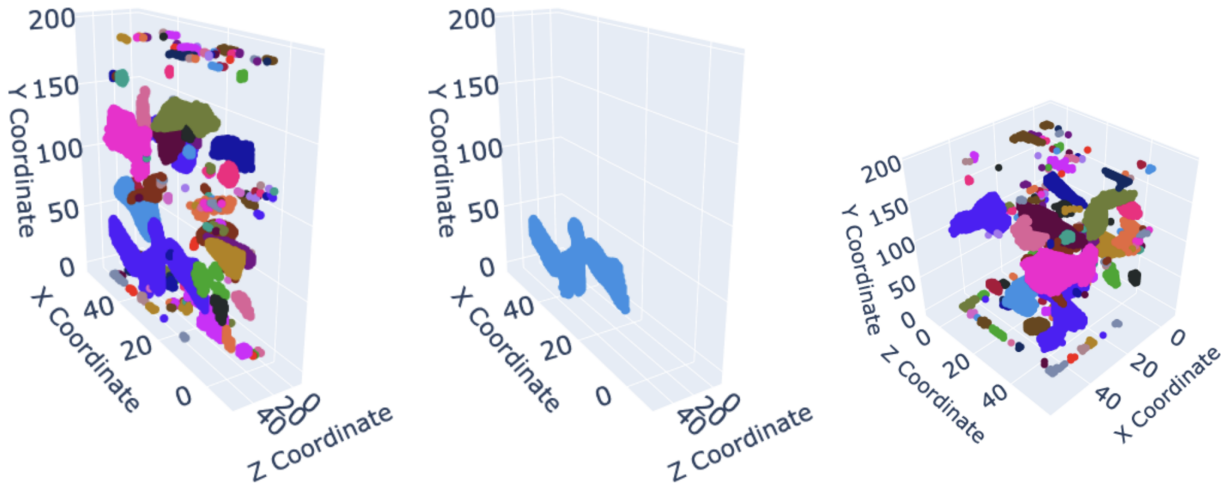
The detection method is straightforward: first, check if a point is a Q-event, then examine its neighbors to see if they are also Q-events. Neighbors that are Q-events are grouped into the same cluster. This principle is applied across the grid to facilitate cluster detection.

To enhance the efficiency of this method, each grid point is treated as an integer tuple (x, y, z) rather than a float tuple (x, y, z). Each space point defined by an integer triplet allows the use of scipy's label function to find connected components (clusters). Therefore, in the visualization, the intervals of (x, y, z) will not be the usual ones but simply indices ranging from 0 to 47 for X and Z, and from 0 to 191 for Y.

3.2.2 Clustering Results

The same time frame, $t = 901.7$, was used for the cluster detection analysis. The algorithm identified 146 clusters for $H = 2$. For the future of this proect, this identification allows for a detailed analysis of how the DRL agent modifies each Q-event and enhances the understanding of its strategy for flow laminarization.

The visualization of these detections can be seen in Figure 3.2. Each cluster is represented by a different, arbitrary color.



(A) Global view. (B) Isolation of cluster 57. (C) Zoom on a cluster.

Figure 3.2: Visualization of the cluster detection algorithm at time $t = 901.7$ for $H = 2$. (A) Global view showing all 146 detected clusters, highlighting the variation in their shapes and sizes. (B) Isolation of cluster 57, allowing for detailed examination of its structure. (C) Zoomed-in view on a cluster, demonstrating the flexibility of the visualization program, including zooming and adjusting viewing angles.

In Figure 3.2A, all 146 clusters are shown. It can be observed that their shapes vary significantly, ranging from nearly a single grid point to much larger and more varied volumes. In the next figure, Figure 3.2B, only the cluster labeled as 57 by the program is displayed. The visualization program allows for the isolation of each cluster to study their size and shape in detail. Finally, Figure 3.2C demonstrates the flexibility of the visualization program, which allows users to zoom, change ratios, and adjust the viewing angle directly within the interface. For hands-on experience, visit the project's GitHub repository at: https://github.com/corentinprados/Q_event_DRL_control.git.

3.3 Percolation Analysis

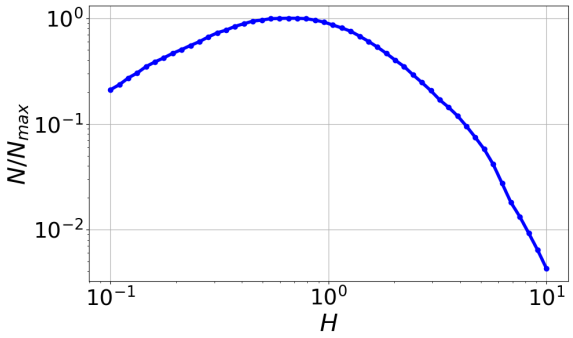
3.3.1 Methodology

As previously discussed, the threshold parameter H significantly impacts the number and volume of Q-event clusters. To determine this parameter optimally, a percolation diagram is used, which plots the number of clusters as a function of H as explained in Section 1.3. This diagram helps identify the optimal H value that balances the detection of small, strong events and larger, more distributed ones.

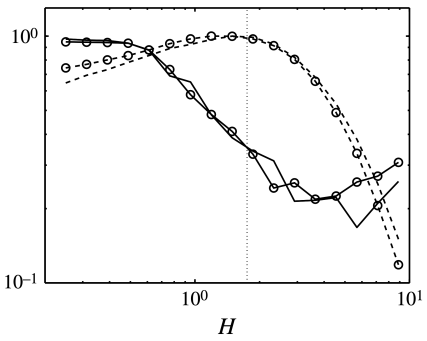
3.3.2 Results and Comparison

To limit computational cost and avoid numerical artifacts, the percolation diagram was created by studying only the truncated interval from $t_i = 895.2$ to $t_f = 935.3$. The value N is the average number of detected clusters over the interval for a fixed H. For good logarithmic distribution, H was chosen logarithmically equidistant between 0.1 and 10, with 50 values tested.

In Figure 3.3A, the percolation diagram is shown. The maximum ratio N/N_{max} is achieved with $H_{max} = 0.66$ and $N_{max} = 364$.



(A) Percolation diagram on the truncated interval $t_i = 895.2$ to $t_f = 935.3$.



(B) Percolation diagram from the literature [44].

Figure 3.3: Comparison of percolation diagrams. Plotting these curves on the same graph would be ideal, but the literature data were not accessible. (A) The curve shows a bell shape but with discrepancies, such as the maximum occurring at a slightly lower H value. (B) ——, Volume ratio of the largest object to the one of all identified objects; – – –, N/N_{max} ratio of the number of identified objects to the maximum number of objects. Lines without symbols correspond to $Re_\tau = 934$ [45], and those with symbols correspond to $Re_\tau = 2003$ [46].

The percolation diagram shows a bell shape similar to the literature (Figure 3.3B), with a slightly increasing curve from 0.1 to 1 and a steeper decreasing curve from 1 to 10. However, the values do not align with those in Lozano-Duran *et al.*. The maximum appears at a slightly lower H value, and the other values differ.

These discrepancies may stem from the lower Reynolds number of the simulation ($Re_\tau = 125$) compared to the literature ($Re_\tau = 934$ [45] and $Re_\tau = 2003$ [46]). Additionally, the simulation is inaccurate. Replotting with a more accurate simulation is needed for deeper analysis.

3.4 Discussion on Q-events Analysis

Despite inaccuracies in the simulation results, this section has provided valuable insights into Q-events, essential for the project's next phase involving DRL implementation. The tools to detect Q-events, identify clusters, and study their number, shape, and volume are now available. Additionally, the percolation diagram tool helps determine the optimal H.

Although the volumetric aspect of the clusters was not studied here, it would be a valuable addition. The strength of clusters, i.e., up to what H they remain clusters, was also not examined. For the DRL algorithm, this might be more important. These aspects could be explored in future studies.

The concept of quadrant analysis was not addressed here, as it is more relevant for post-processing after applying reinforcement learning to the Alya simulation, using Q-events.

4 DRL with Dedalus and Stable-Baselines3

4.1 Introduction and Topic Selection

This section explains why I changed the project's direction. I apologize if it seems uninteresting, but it is necessary for understanding the approach. Thank you for your understanding and taking the time to read it.

4.1.1 Challenges and Initial Difficulties

The goal of this internship was to gain practical knowledge in Machine Learning through a project. To build a foundation, I studied Andrew Ng's courses for two months, which gave me the basics but not practical application skills. I relied on the internship project for this hands-on experience.

However, I faced significant challenges. I had to work with an unfinished, uncommented codebase of several thousand lines, which was overly ambitious for a beginner. Supervision was minimal, with the supervisor nearly inaccessible and colleagues always overbooked, making it impossible to accomplish the task in such a short time.

The code was only adapted for Alya simulations, which I struggled with, and it was suitable only for 3D simulations. Attempting a DRL project in 3D without a powerful computer was impractical within the two-month timeframe. Combining DRL and CFD in 3D is not optimal for a beginner due to high computation costs and the need for numerous small tests to understand the code.

The Alya code was written in Fortran, while the DRL code used TensorFlow with Python, requiring cross-language communication skills. Additionally, the TensorFlow code was uncommented and used MARL (Multi-Agent Reinforcement Learning), which was too complex for a first reinforcement learning project.

With only two months left, including report writing, this project was unrealistic, as I didn't feel confident enough to complete it. Although the code used parallelization to mitigate time issues slightly, it added another layer of difficulty, especially as it was directly tied to the MARL principle, which was beyond my expertise.

4.1.2 Decision to Change Direction

Given the overall situation and difficulties, I explained to the team that I didn't feel capable of completing the project. My objective was to learn as much as possible about Machine Learning and undertake a DRL project to apply it. However, the proposed project was beyond my level.

I took the initiative to change the project choice. I explained that my goal was not to abandon the project but to take a more reasonable step to later successfully complete it. My aim was to learn as much as possible through a project at my level and then apply the knowledge to the initial project, which involved using DRL with Alya to minimize Q-events.

The team accepted my decision and engaged others to divide the project into more manageable parts. By utilizing my work on Alya and Q-events, particularly through GitHub, the newcomers could quickly progress to the Machine Learning stage. This approach allowed them to bypass initial barriers and assemble the project's components—CFD, turbulence, and Machine Learning—more efficiently. I was thanked for my contribution, which provided a smoother path for the newcomers.

4.1.3 Strategic Simplification of the New Project

To maximize my learning time for Machine Learning, I made several strategic choices to simplify the project.

Firstly, I removed all difficulties unrelated to reinforcement learning. I chose Dedalus, a CFD framework I am proficient in, with good documentation and community support, which uses Python. This eliminated the problem of code communication. Dedalus also employs a spectral method, avoiding the need for a mesh like with Gmsh and Alya.

I minimized computation time by running 2D simulations instead of 3D, thus avoiding the complexity of parallelization, which I tried but failed to implement. I simplified the Machine Learning aspect by working with a single agent and using a user-friendly framework, Stable-Baselines3, designed for reinforcement learning and coded in PyTorch.

New difficulties arose. Minimizing turbulence in 2D is challenging because true 2D turbulence occurs only at very high Reynolds numbers, increasing time and computation costs [52]. After many tests, I concluded that it wasn't feasible or of much interest. Since the goal was to learn reinforcement learning, not advance the specific project questions, I completely changed the topic, as explained in the next section.

4.2 Computational Fluid Dynamics Overview

4.2.1 Framework of the Simulation

The fundamental equations for the CFC case described in Section 1.2 are revisited, focusing on the two-dimensional (x, y) scenario. The same initial and boundary conditions are applied. However, in this context, a pressure-driven flow is chosen, and non-dimensionalization is performed using $Re = \frac{1}{\nu}$.

The pressure gradient is defined as $p = -2\nu x$. Incorporating this pressure gradient into the Navier-Stokes equations and expressing them in terms of the Reynolds number (Re), the governing equation becomes:

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{U} + \frac{2}{Re} \mathbf{e}_x$$

Here, the pressure p is the classical pressure P divided by the mass density ρ, i.e., $p = \frac{P}{\rho}$.

4.2.2 Stable States and Reynolds Number in 2D Flows

The behavior of 2D flows can be categorized based on the Reynolds number:

At low Reynolds numbers, below $Re = 5400$, the flow remains stable and laminar. The velocity profile is $\mathbf{U}_{base}(y) = (1 - y^2, 0)$, named the parabolic state.

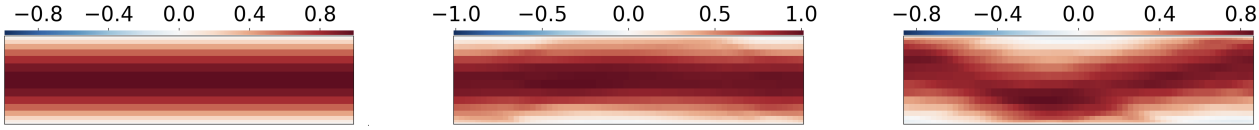
During the transition phase, approximately $Re \approx 5400 - 7000$, the flow starts to oscillate periodically. This phase is characterized by a limit cycle where the wall stress oscillates periodically.

At high Reynolds numbers, $Re \approx 7000$ and above, before turbulence sets in, the parabolic state becomes metastable. It transitions into a new stable, nearly periodic state, forming a wave-like oscillating pattern, named the oscillating state [53].

For a better understanding of the parabolic and oscillating states, snapshots from the simulation using the described parameters can be found in Figure 4.1. The parabolic state, used as the initial condition for the simulation, is shown on the left. The oscillating state is depicted on the right. The form of the transition state for approximately $Re \approx 5400 - 7000$ is not shown.

4.2.3 Simulation Setup and Results

The simulation was set up with the following parameters: a Reynolds number $Re = 10^4$ was used. The domain dimensions were set to a height of $Ly = 1$ in the y-direction and a length of $Lx = 4$ in the x-direction. The resolution in the y-direction was $Ny = 16$, while in the x-direction it was $Nx = 64$. For the grid, a Fourier basis was chosen in the x-direction and a Chebyshev basis in the y-direction. The dealiasing factor was set to 3/2. The initial condition noise amplitude was $\alpha = 10^{-3}$, the CFL safety factor was 0.1, and the time stepping method used was RK222. The Figure 4.1 illustrates three key phases of the simulation.



(A) Parabolic state as initial condition, $t = 0$.

(B) Transition state between parabolic and oscillating, $t = 529$.

(C) Oscillating state form, $t = 950$, stable state.

Figure 4.1: Simulation results at a Reynolds number $Re = 10,000$ for a duration of 10,000 numerical time units for the streamwise velocity U_x. From $t = 0$ to approximately $t = 80$, the parabolic state remains stable; from around $t = 80$ to $t = 600$, the flow destabilizes and transitions; from $t = 600$ onwards, the new oscillating stable state is established, with no further changes in the flow.

4.2.4 Objective of Deep Reinforcement Learning

Based on the aforementioned theory, the goal of the RL program is to transition the system from the oscillating stable state at $Re = 10,000$ back to the parabolic state through agent actions. The agent achieves this by periodically injecting a jet into the flow. The details of this RL program and its implementation are elaborated in the following section.

4.3 Integration of RL and CFD for Flow Control Optimization

This section aims to demonstrate the application of RL in conjunction with CFD, utilizing the Stable-Baselines3 and Dedalus frameworks. The objective is to employ RL to control the simulated flow dynamics. The complete project is available on GitHub and the code detailed is included in the Appendix **??**, however, it is not necessary to refer to that for this section. Each critical component of the RL setup is discussed here, detailing their implementation and role in the overall system. It is beneficial to keep Figure 1.3 from Section 1.1 in mind, as it illustrates how RL functions and aids in understanding each step of the process. The times used in this section are time units as output by the Dedalus simulation, based on the formulation of the equations.

4.3.1 Defining the Environment

Firstly, the environment must be defined. The environment, denoted as the `FlowControlEnv` class, serves as the platform where the agent interacts with the flow dynamics. This class inherits from the `gym.Env` class provided by OpenAI Gym, a toolkit for developing and comparing reinforcement learning algorithms. It is responsible for defining the state and action spaces and for managing the flow simulation using Dedalus.

Action Space

The action space represents the agent's control over the environment. In this setup, it is defined as a single continuous parameter, α, within the range $[-\alpha_{\max}, \alpha_{\max}]$. The parameter $\alpha_{\max}$ was chosen based on comparisons with literature on DRL applied to CFD and through a series of trial and error experiments. This ensures the action is significant enough to influence the flow while being sufficiently small to avoid destabilizing it.

Observation Space

The observation space is essential for the agent to perceive the environment's state. It consists of a grid representing the two-dimensional velocity fields extracted from the Dedalus simulation: $U(x, y, t)$, the streamwise velocity, and $V(x, y, t)$, the wall-normal velocity at the time of the step. This continuous space, with dimensions $(N_x \times \text{dealias}, N_y \times \text{dealias}, 2)$, includes the dealiased grid points in the x and y directions and the two velocity components.

Dedalus Simulation Initialization

Initialization of the Dedalus simulation involves setting up the computational domain, defining spatial bases, and initializing velocity and pressure fields. The problem is formulated as an initial value problem (IVP) with the Navier-Stokes equations defined in the Section 4.2, incorporating boundary conditions and pressure gauge conditions. The simulation employs a dealiased Fourier-Chebyshev basis for high accuracy. The solver utilizes an adaptive timestepper with a CFL condition to ensure numerical stability.

Clarification of Terms

A simulation step is a single advancement in time where the environment's state is updated. Agent action moments occur periodically every $T = 1$. An episode starts from a checkpoint at $t_0 = 950$ and runs for $\Delta t = 70$, concluding with a restart from the checkpoint. To handle divergence, the episode restarts if the simulation diverges. This structure provides the agent with multiple opportunities to learn and optimize actions within a controlled and repeatable timeframe.

4.3.2 Agent's Decision-Making Process

The agent processes the observed state s through its neural network to determine an action a, directly impacting the environment and creating a feedback loop where the new state is observed. The agent's action is periodic, activating a jet every $T = 1$, and follows a sigmoid-like function rather than a simple Heaviside step function:

$$\alpha(t) = \alpha_{\text{agent}}(t_i) \cdot H(t - (t_i - t_0)) \cdot \exp\left(-\frac{1}{k \cdot (t - (t_i - t_0))}\right),$$

where $\alpha_{\text{agent}}(t_i)$ is the action chosen at time t_i, H is the Heaviside function, k is a growth factor, $t_0 = 950$ is the initial checkpoint time, and t is the current time.

Initially, the objective was to position the jet at the walls, acting on the normal velocity component. However, due to limitations in Dedalus, the jet's direction was retained but positioned within the flow at a specific location to avoid imposing any particular symmetry. Experimentally, it was found that placing the jet symmetrically resulted in its effect being nullified by the simulation. The location is approximately in the lower right region, precisely defined as:

$$G(x, z) = \alpha(t) \cdot (H(x - 2) - H(x - 3)) \cdot (H(z + 0.5) - H(z)).$$

4.3.3 Maximizing the Reward

The action chosen by the agent based on the observed state aims to maximize the reward. Mathematically, $\Delta N(t)$ is defined as:

$$\Delta N(t) = \|\underline{U}(x, y, t) - \underline{U}_{\text{parabolic}}(y)\|_2,$$

where $\underline{U}(x, y, t)$ represents the current state velocity field and $\underline{U}_{\text{parabolic}}(y)$ represents the parabolic state velocity field defined in the previous section. This metric quantifies the difference between the current state and the desired parabolic state. Let ΔN_0 denote $\Delta N(t_0)$.

Initially, the reward function was chosen to be a simple linear function:

$$R_{\text{linear}} = \text{clip}(1 - \Delta N/\Delta N_0, -1, 1).$$

The goal was for the reward to reach one when the objective was achieved ($\Delta N = 0$) and to have a minimal, increasingly negative value as ΔN grew. To prevent numerical instability, this function was clipped to the interval $[-1, 1]$.

Ultimately, it was realized that this reward function was too simplistic to capture the full range of desired outcomes. Therefore, the function was scaled by a factor of three to maintain its significance. Additionally, the following components were added:

- **Exponential Base Reward:** $R_{\text{exp}} = 3 \cdot \exp(-\Delta N)$. This exponential term encourages exploration that reduces ΔN.
- **Improvement Reward:** $R_{\text{improve}} = \text{clip}((\max(0, \Delta N_{\text{prev}} - \Delta N))^2, 0, 1)$. This reward promotes changes that move towards the objective at each step, where ΔN_{prev} is the ΔN calculated in the previous step.
- **Stability Reward:** $R_{\text{stability}} = 1$ if $\Delta N < 1$ else 0. This provides a fixed reward for maintaining deviations within a low threshold, promoting stability.
- **Control Penalty:** $P_{\text{control}} = 25 \cdot |\alpha_{\text{agent}}(t_i)| \cdot (1 - \frac{1}{\text{episode steps}}) + 3 \cdot |\alpha_{\text{agent}}(t_i)| + \text{clip}(\Delta N/\Delta N_0, 0, 1)$. This penalizes larger control actions, ensuring the agent uses minimal intervention to achieve its goals. Large values for α do not promote flow stability, especially when the flow has already been stabilized.
- **Terminal Rewards and Penalties:** At the end of each episode, additional rewards or penalties are applied based on the final deviation from the parabolic state. These incentives ensure that the agent aims to reach and maintain the desired state by the end of each episode.

The amplitude of each reward component has been carefully chosen to impose a hierarchy between the most important rewards and those that stabilize the agent's actions towards the end of training. While maintaining a reasonable range of possible values for the reward, specifically around [-10, 10].

4.3.4 Neural Network Update

The neural network associated with the agent is updated at each step. This update process adjusts the network's parameters to improve its performance in selecting actions that maximize the cumulative reward over time. In this implementation, the Proximal Policy Optimization (PPO) method is used. PPO involves collecting a set of trajectories by running the current policy in the environment, calculating the advantage estimates for each state-action pair, and updating the policy by minimizing the PPO objective. This method ensures that the new policy is not too far from the old policy, thereby maintaining stability during training.

4.3.5 Learning Objective

By continuously looping through this process, the agent learns how to achieve the objective. The iterative cycle of observing the state, deciding and applying actions, receiving rewards, and updating the neural network enables the agent to improve its strategy for controlling the flow dynamics effectively.

4.4 Results

4.4.1 Experiment Parameters

After several iterations of trial and error, the model was trained using specific hyperparameters for an extended period. The learning rates of 0.001 and 0.0001 controlled the step size in optimization, with smaller rates offering precision and larger rates providing speed. Batch sizes of 32 and 64 determined the number of training samples per iteration, balancing accuracy and computational cost. The discount factor γ, set at 0.95 and 0.99, balanced immediate and future rewards. These configurations resulted in eight parameter triplets, each executed for $N_{steps} = 4096 \times 15000$ timesteps with saves every 1000 steps. Wandb facilitated real-time data saving. An entropy coefficient, `ent_coef`, set at 0.01, was included to encourage exploration and stabilize learning. The training lasted one week on the office workstation, utilizing separate cores for each triplet.

4.4.2 Training Results

Mean Reward per Episode

The mean rewards per episode are shown in Figure 4.2. The programs did not all stop at the same number of timesteps, and none reached the target N_{steps}. The mean reward per episode fluctuated between -4 and -4.8, the minimum possible reward. Notably, the sessions that stopped earliest learned the fastest, with the parameter set (lr = 0.0001, bz = 32, $\gamma = 0.99$) quickly rising from -4.8 to around -4.5 in the smoothed curve. Other simulations failed to rise from -4.7.

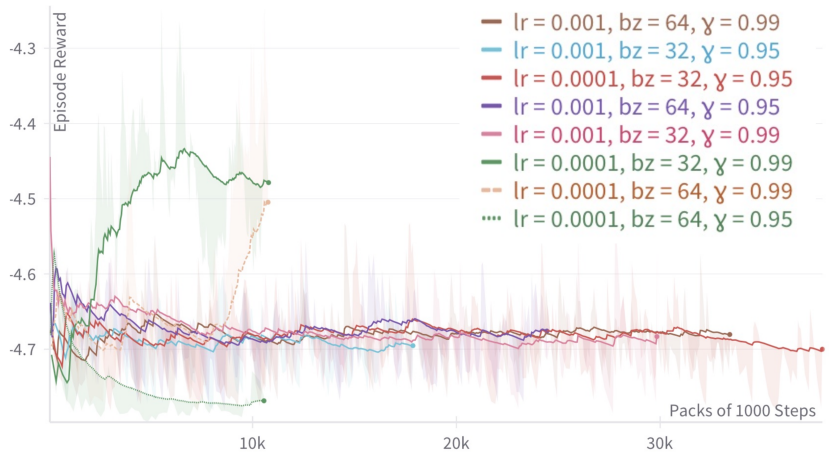


Figure 4.2: Smoothed mean reward per episode for each parameter triplet is shown. Unexpected training stops prevented reaching a reward close to one. The non-smoothed version is available in the Appendix **??**.

The program stops due to [nan] values in one of the model tensors, an issue still under investigation. Restarting sessions did not resolve the problem, as the [nan] values persisted and caused crashes. However, some parameter sets learned effectively, indicating the algorithm is sound but needs refinement.

To improve stability, several adjustments can be made. Reducing the force applied in simulations could help avoid large variations. A more progressive reward structure might prevent rapid changes in the learning algorithm. Making jet injections more gradual and increasing simulation resolution could also enhance stability. Lastly, adding more layers to the neural network could improve its capacity to learn complex patterns. Previous attempts to reduce input parameters or increase output parameters with more jets did not succeed, indicating further refinement is needed.

Short-Term Statistics

Other statistics saved every 1000 timesteps provide insight into the training stops. These statistics, shown in Figure 4.3, correspond to agent actions rather than rewards. For the first five parameter sets, the mean and standard deviation values initially hover around random decisions and then diverge dramatically. This divergence, which is not visible in the log, likely causes the training stops, but its cause remains unidentified.

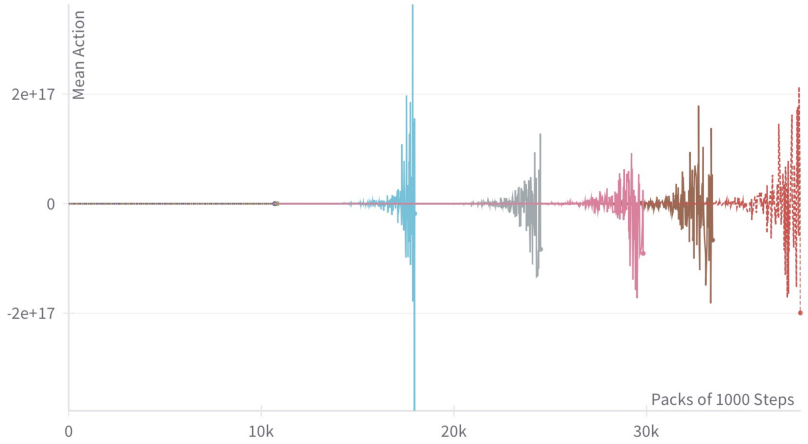


Figure 4.3: Statistics of actions over a period of 1000 timesteps (mean) are shown. The actions escalate to absurd values, leading to training divergence. The standard deviation is available in the Appendix **??**.

Episode Statistics

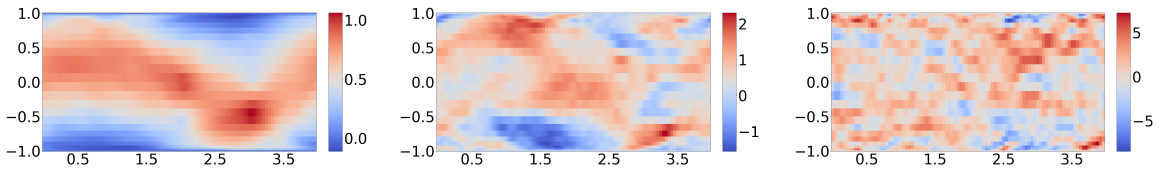
The overall averages are 81.38 actions per episode, 46838 steps per episode, and a duration of 724.57 seconds per episode. A detailed summary of key statistics for each parameter triplet is available in the Appendix **??**.

The statistics indicate that the average number of actions per episode is around 80, which is relatively low for a reinforcement learning program, especially considering approximately 50,000 steps per episode. This significant discrepancy is typical in reinforcement learning with CFD, where actions must be spaced out to prevent simulation divergence and to observe the effects of the agent's actions. However, the divergence issue complicates achieving a large number of episodes.

This suggests reconsidering how neural network updates are coded. Currently, rewards are calculated at each timestep, but it might be more effective to calculate rewards only when the agent's actions are implemented and for a specific duration. These are potential directions to explore in the coming weeks.

4.4.3 Agent's Method Analysis

By examining images of the agent's actions to stabilize the flow (Figure 4.4), its methods and shortcomings can be better understood, although these are preliminary insights as the training has not yet yielded satisfactory results. The model with the best average reward (lr = 0.0001, bz = 32, $\gamma = 0.99$) was used. Initially, the agent effectively injects a jet to attenuate the wave, aiming to destroy the oscillating state and reestablish a parabolic state. However, subsequent actions destabilize the flow excessively, leading to neither the initial nor the desired state, with actions consistently reaching the maximum and minimum values $\alpha_{\max}$ and $-\alpha_{\max}$. It appears that these excessively strong actions become residuals that the agent cannot manage, as evidenced by the last figure where locally very high velocities are observed.



(A) $t = 1.9$ (B) $t = 24.2$ (C) $t = 70$

Figure 4.4: Application of the model with parameter set (lr = 0.0001, bz = 32, $\gamma = 0.99$). Initial jet injection attenuates the wave, but subsequent actions destabilize the flow.

Given these results, penalizing strong jet actions more heavily during episodes and allowing more time for learning could be beneficial. However, the divergence issue hinders progress. Notably, a week of training provides minimal learning, highlighting the significant computational cost of reinforcement learning with CFD.

Conclusion

Conclusion on the CFD part

Alya is effective for fluid dynamics simulations but has a steep learning curve due to insufficient documentation and a small user community, causing several issues during this project. A team member is now focused on mastering Alya to integrate it with machine learning, which is promising for future applications like aviation. SOD2D, an evolution of Alya, is gaining traction at KTH and is expected to replace Alya.

My experience with Alya was frustrating due to the extensive learning time and lack of powerful simulation tools. While I have paved the way for future colleagues, I am still far from using Alya efficiently. However, this exploration has added valuable tools to my CFD toolkit, beneficial for my future research, including my PhD.

Conclusion on the Q-events part

The concept of Q-events has become much clearer for the team, especially for myself and the newer members, and shows significant promise in reducing turbulence in flows. Team members are now specializing in this area to gain a deeper understanding of the coherent structures that arise from Q-events.

Understanding these coherent structures has enriched my knowledge of turbulence, which will be useful in my future research and thesis. Additionally, my efforts will provide future colleagues with visualization and translation tools for Alya in the context of Q-events.

Conclusion on the DRL part

This section, although not directly aligned with the initial project, was where I learned the most. The theoretical foundation I built at the beginning of the internship was essential for understanding machine learning, but it did not prepare me to lead a project. Conversely, the extensive hours spent on this part significantly enhanced my skills. I now understand all the critical points necessary to create a reinforcement learning program and how to manage such a project, especially with CFD. I still have much to learn and am not yet an expert, but I am now better prepared to contribute to the initial project proposed by my supervisor.

I do not have conclusive outcomes yet, and it is uncertain if achieving my theoretical goals was possible. However, my primary objective was to learn, and in that respect, the project has been very successful. Despite the lack of concrete results, I am satisfied with the knowledge gained.

Personal Note on the Internship

My goals in coming to Sweden were to improve my English and add machine learning to my research toolkit. Although this internship may not lead to a publication, it has fully met its purpose of educating me. According to me, given the power of artificial intelligence, it is essential for young scientists to have at least a basic understanding of machine learning. I am glad to have taken this step to stay current, particularly in CFD.

I am very pleased to have completed my internship at KTH, which exposed me to a different supervisory method than in France. Here, the structure is more hierarchical: doctoral positions communicate with PhD students, who communicate with master's students, who then interact with undergraduates. This creates a more solitary work environment for master's students. Unlike in France, where I worked closely with the researcher, at KTH I mainly interacted with my supervisor during meetings or progress presentations. This experience also allowed me to supervise two undergraduate students.

Project Outlook

The project conducted during this internship, which will continue for a few more weeks, is still in its early stages. The significant workload involves combining various fields like physics, CFD, recent turbulence results, and machine learning, meaning my work has primarily laid the groundwork rather than brought the project to completion. This is frustrating but an important step in the project's development.

The recent division of work among several colleagues will enable faster progress. The integration of Alya with reinforcement learning to minimize Q-events will soon be implemented, which will be my focus for the remaining three weeks of the internship. The application of machine learning to fluid mechanics is still burgeoning. This project holds great promise for the future of this field.

Bibliography

[1] International Energy Agency. Largest end uses of energy by sector in selected iea countries, 2019, 2021. IEA, Paris https://www.iea.org/data-and-statistics/charts/largest-end-uses-of-energy-by-sector-in-selected-iea-countries-2019, Licence: CC BY 4.0.

[2] R.M. Wood. Aerodynamic drag and drag reduction: Energy and energy savings. *41st Aerospace Sciences Meeting and Exhibit*, 2003.

[3] Brian Dean and Bharat Bhushan. Shark-skin surfaces for fluid-drag reduction in turbulent flow: A review. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368: 4775–4806, 2010. doi: 10.1098/rsta.2010.0201.

[4] Mohamed Gad-el Hak. Modern developments in flow control. *Applied Mechanics Reviews*, 49:365–379, 1996. doi: 10.1115/1.3101931.

[5] Steven L. Brunton and Bernd R. Noack. Closed-loop turbulence control: Progress and challenges. *Applied Mechanics Reviews*, 67(5):050801, 2015. doi: 10.1115/1.4031175. URL https://doi.org/10.1115/1.4031175.

[6] Jens Pfeiffer and Rudibert King. Robust control of drag and lateral dynamic response for road vehicles exposed to cross-wind gusts. *Experiments in Fluids*, 59, 2018. doi: 10.1007/s00348-017-2479-7.

[7] Fabio Pino, Lorenzo Schena, Jean Rabault, and Miguel A. Mendez. Comparative analysis of machine learning methods for active flow control. *Journal of Fluid Mechanics*, 958, 2023. doi: 10.1017/jfm.2023.76.

[8] Thomas Duriez, Steven Brunton, and Bernd Noack. *Machine Learning Control – Taming Nonlinear Dynamics and Turbulence*, volume 116. Springer, 2016. ISBN 978-3-319-40623-7. doi: 10.1007/978-3-319-40624-4.

[9] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Réglade, and Nicolas Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302, 2019. doi: 10.1017/jfm.2019.62.

[10] Luca Guastoni, Jean Rabault, Philipp Schlatter, Hossein Azizpour, and Ricardo Vinuesa. Deep reinforcement learning for turbulent drag reduction in channel flows, 2023.

[11] Andres Cremades, Sergio Hoyas, Rahul Deshpande, Pedro Quintero, Martin Lellep, Will Junghoon Lee, Jason Monty, Nicholas Hutchins, Moritz Linkmann, Ivan Marusic, and Ricardo Vinuesa. Identifying regions of importance in wall-bounded turbulence through explainable deep learning, 2024.

[12] Andrew Ng. Machine learning specialization, 2022. URL https://www.coursera.org/specializations/machine-learning-introduction. Coursera, accessed February 2024.

[13] Steven L. Brunton. Machine learning meets control theory, 2021. URL https://doi.org/10.52843/cassyni.x2t0sp. Reinforcement learning, 2021/02/12.

[14] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

[15] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52(1):477–508, jan 2020. ISSN 1545-4479. doi: 10.1146/annurev-fluid-010719-060214. URL http://dx.doi.org/10.1146/annurev-fluid-010719-060214.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] Takaaki Murata, Kai Fukami, and Koji Fukagata. Nonlinear mode decomposition with convolutional neural networks for fluid dynamics. *Journal of Fluid Mechanics*, 882:A13, 2020. doi: 10.1017/jfm.2019.822.

[18] Isabel Scherl, Benjamin Strom, Jessica K. Shang, Owen Williams, Brian L. Polagye, and Steven L. Brunton. Robust principal component analysis for modal decomposition of corrupt fluid flows, 2019.

[19] Kuan Zhang, Haoji Hu, Kenneth Philbrick, Gian Marco Conte, Joseph D. Sobek, Pouria Rouzrokh, and Bradley J. Erickson. Soup-gan: Super-resolution mri using generative adversarial networks, 2021.

[20] N. Benjamin Erichson, Lionel Mathelin, Zhewei Yao, Steven L. Brunton, Michael W. Mahoney, and J. Nathan Kutz. Shallow neural networks for fluid flow reconstruction with limited sensors. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2238):20200097, jun 2020. ISSN 1471-2946. doi: 10.1098/rspa.2020.0097. URL http://dx.doi.org/10.1098/rspa.2020.0097.

[21] Jared L. Callaham, Kazuki Maeda, and Steven L. Brunton. Robust flow reconstruction from limited measurements via sparse representation. *Physical Review Fluids*, 4(10), oct 2019. ISSN 2469-990X. doi: 10.1103/physrevfluids.4.103907. URL http://dx.doi.org/10.1103/PhysRevFluids.4.103907.

[22] Ricardo Vinuesa and Steven L. Brunton. Enhancing computational fluid dynamics with machine learning. *Nature Computational Science*, 2(6):358–366, jun 2022. ISSN 2662-8457. doi: 10.1038/s43588-022-00264-7. URL http://dx.doi.org/10.1038/s43588-022-00264-7.

[23] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31): 15344–15349, jul 2019. ISSN 1091-6490. doi: 10.1073/pnas.1814058116. URL http://dx.doi.org/10.1073/pnas.1814058116.

[24] Andrea Beck, David Flad, and Claus-Dieter Munz. Deep neural networks for data-driven les closure models. *Journal of Computational Physics*, 398:108910, dec 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2019.108910. URL http://dx.doi.org/10.1016/j.jcp.2019.108910.

[25] Mattia Cenedese, Joar Axås, Bastian Bäuerlein, Kerstin Avila, and George Haller. Data-driven modeling and prediction of non-linearizable dynamics via spectral submanifolds. *Nature Communications*, 13 (1), feb 2022. ISSN 2041-1723. doi: 10.1038/s41467-022-28518-y. URL http://dx.doi.org/10.1038/s41467-022-28518-y.

[26] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[27] Guido Novati, Hadrien L. de Laroussilhe, and Petros Koumoutsakos. Automating turbulence modelling by multi-agent reinforcement learning. *Nature Machine Intelligence*, 3(1):87–96, 2021.

[28] Siddhartha Verma, Guido Novati, and Petros Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23): 5849–5854, may 2018. ISSN 1091-6490. doi: 10.1073/pnas.1800923115. URL http://dx.doi.org/10.1073/pnas.1800923115.

[29] C. Bauer, Y. Sakai, and M. Uhlmann. Direct numerical simulation of turbulent open channel flow: Streamwise turbulence intensity scaling and its relation to large-scale coherent motions. In *Proceedings of the 10th iTi Conference on Turbulence*, Italy, 2023. International Conference on Turbulence. URL https://www.iti-conference.com/proceedings/2023. DNS data of turbulent closed and open channel flow in boxes of $L_x/h = 12\pi$, $L_z/h = 4\pi$ for $Re_\tau = 200, 400, 600, 900$.

[30] Javier Jiménez and Parviz Moin. The minimal flow unit in near-wall turbulence. *Journal of Fluid Mechanics*, 225:213–240, 1991. doi: 10.1017/S0022112091002033.

[31] Javier Jiménez. Coherent structures in wall-bounded turbulence. *Journal of Fluid Mechanics*, 842, mar 2018. ISSN 1469-7645. doi: 10.1017/jfm.2018.144. URL http://dx.doi.org/10.1017/jfm.2018.144.

[32] Laurent Brodeau, Julien Le Sommer, and Aurélie Albert. ocean-next/enatl60: Material describing the set-up and the assessment of nemo-enatl60 simulations (version v1), 2020. URL https://doi.org/10.5281/zenodo.4032732.

[33] Ronald Adrian. Hairpin vortex organization in wall turbulence. *Physics of Fluids*, 19, 04 2007. doi: 10.1063/1.2717527.

[34] Marco Atzori. *Coherent structures and control in wall-bounded turbulent flows*. PhD thesis, KTH Royal Institute of Technology, 2021.

[35] J. Hunt, Alan Wray, and Parviz Moin. Eddies, streams, and convergence zones in turbulent flows. *Studying Turbulence Using Numerical Simulation Databases*, -1:193–208, 11 1988.

[36] M. S. Chong, A. E. Perry, and B. J. Cantwell. A general classification of three-dimensional flow fields. *Physics of Fluids A*, 2(5):765–777, may 1990. doi: 10.1063/1.857730.

[37] Jinhee Jeong and Fazle Hussain. On the identification of a vortex. *Journal of Fluid Mechanics*, 332:339–363, 01 1995.

[38] Juan C. Del Alamo and Paulo Zandonade. Self-similar vortex clusters in the turbulent logarithmic region. *Journal of Fluid Mechanics*, 561:329–358, 08 2006. doi: 10.1017/S0022112006000814.

[39] Andres Cremades, Sergio Hoyas, Rahul Deshpande, Pedro Quintero, Martin Lellep, Will Junghoon Lee, Jason Monty, Nicholas Hutchins, Moritz Linkmann, Ivan Marusic, and Ricardo Vinuesa. Identifying regions of importance in wall-bounded turbulence through explainable deep learning, 2024.

[40] S. S. Lu and W. W. Willmarth. Measurements of the structure of the reynolds stress in a turbulent boundary layer. *Journal of Fluid Mechanics*, 60(3):481–511, 1973. doi: 10.1017/S0022112073000315.

[41] Luca Guastoni. *Time, space and control: deep-learning applications to turbulent flows*. PhD thesis, KTH Royal Institute of Technology, 2023.

[42] Ryuichi Nagaosa and Robert Handler. Statistical analysis of coherent vortices near a free surface in a fully developed turbulence. *Physics of Fluids*, 15:375–394, 02 2003. doi: 10.1063/1.1533071.

[43] D. G. Bogard and W. G. Tiederman. Burst detection with single-point velocity measurements. *Journal of Fluid Mechanics*, 162:389–413, 1986. doi: 10.1017/S0022112086002094.

[44] Adrián Lozano-Durán, Oscar Flores, and Javier Jiménez. The three-dimensional structure of momentum transfer in turbulent channels. *Journal of Fluid Mechanics*, 694:100–130, 2012. doi: 10.1017/jfm.2011.524.

[45] Juan C. Del Álamo, Javier Jiménez, Paulo Zandonade, and Robert D. Moser. Scaling of the energy spectra of turbulent channels. *Journal of Fluid Mechanics*, 500:135–144, 2004. doi: 10.1017/S002211200300733X.

[46] Javier Hoyas, Sergio et Jiménez. Scaling of the velocity fluctuations in turbulent channels up to re=2003. *Physics of Fluids*, 18(1):011702, 01 2006. ISSN 1070-6631. doi: 10.1063/1.2162185. URL https://doi.org/10.1063/1.2162185.

[47] Adrian Lozano-Duran, Oscar Flores, and Javier Jiménez. The three-dimensional structure of momentum transfer in turbulent channels. *Journal of Fluid Mechanics*, 694:100–130, 03 2012. doi: 10.1017/jfm.2011.524.

[48] Marco Atzori, Ricardo Vinuesa, and Philipp Schlatter. Control effects on coherent structures in a non-uniform adverse-pressure-gradient boundary layer. *International Journal of Heat and Fluid Flow*, 97: 109036, 2022. ISSN 0142-727X. doi: https://doi.org/10.1016/j.ijheatfluidflow.2022.109036.

[49] Pol Suárez, Francisco Alcántara-Ávila, Arnau Miró, Jean Rabault, Bernat Font, Oriol Lehmkuhl, and R. Vinuesa. Active flow control for three-dimensional cylinders through deep reinforcement learning, 2023.

[50] Javier Hoyas, Sergio et Jiménez. Reynolds number effects on the reynolds-stress budgets in turbulent channels. *Physics of Fluids*, 20(10):101511, 10 2008.

[51] Myoungkyu Lee and Robert D. Moser. Direct numerical simulation of turbulent channel flow up to $Re_\tau \approx 5200$. *Journal of Fluid Mechanics*, 774:395–415, 2015. doi: 10.1017/jfm.2015.268.

[52] Igor Vigdorovich. The spectrum of decaying 2d self-similar turbulence. *Doklady Physics*, 64:176–180, 04 2019. doi: 10.1134/S1028335819040086.

[53] Javier Jiménez. Transition to turbulence in two-dimensional poiseuille flow. *Journal of Fluid Mechanics*, 218:265–297, 1990. doi: 10.1017/S0022112090001008.

A DRL with Dedalus and Stable-Baselines3

A.1 Detailed Project Explanation

A.1.1 Project Overview

This project involves training a Proximal Policy Optimization (PPO) model for a flow control environment using the Stable Baselines3 library. The project is structured into several scripts and directories, each serving a specific purpose in the training pipeline. It can foundon the GitHub https://github.com/corentinprados/Q_event_DRL_control.git.

A.1.2 Main Scripts and Functionalities

main_program.py

The `main_program.py` script is responsible for setting up and training the PPO model. Key functionalities include:

- **Importing Required Libraries:** Includes libraries such as `os`, `logging`, `numpy`, and `stable_baselines3`.
- **Defining Helper Functions:**
  - `print_model_architecture(model)`: Prints the architecture of the policy network of the PPO model.
  - `train_environment(restart=False)`: Trains or continues training a PPO model for the flow control environment. This function sets up logging, initializes the environment and model, and trains the model while saving the best version based on performance.

multi_training.py

The `multi_training.py` script is used for multi-process training. Key functionalities include:

- **Importing Required Libraries:** Includes libraries such as `os`, `logging`, `multiprocessing`, and `stable_baselines3`.
- **Defining Helper Functions:**
  - `print_model_architecture(model)`: Prints the architecture of the policy network of the PPO model.
  - `train_environment(config, restart=False)`: Trains or continues training a PPO model for the flow control environment with a specified configuration.

logging_setup.py

The `logging_setup.py` script configures the logging settings for the project. Key functionalities include:

- `setup_logging(log_dir, log_filename="training.log")`: Sets up logging configuration to log to a file. This function ensures the log directory exists, creates the log file, and sets the logging level to `INFO`.

A.1.3 Project Directory Structure

The project directory is organized as follows:

```
my_project_wandb/
 flow_control/
 post_analysis/
 main_program.py
 multi_training.py
 logging_setup.py
 checkpoints_initial_conditions/
 ...
```

Flow Control Environment

The `flow_control` directory contains the implementation of the custom environment used for training the PPO model. It includes environment definitions and callback implementations specific to flow control tasks.

Post Analysis

The `post_analysis` directory contains scripts and notebooks for analyzing the results of the training runs. This may include visualizations, performance metrics, and other analyses.

Model Checkpoints

The `checkpoints_initial_conditions` directory stores initial conditions and checkpoints of the model during training. This allows for resuming training from specific points and comparing different training runs.

Logging and Monitoring

The project utilizes logging and monitoring tools, configured via the `logging_setup.py` script, to keep track of training progress, model performance, and any issues encountered during training.

A.2 Detailed Explanation of `flow_control_env.py`

This section provides a detailed, progressive, and pedagogical explanation of the `flow_control_env.py` file, which implements a custom environment for flow control using Dedalus simulations and the OpenAI Gym interface. Each function is explained with its intellectual construction and its necessity in building a reinforcement learning program.

A.2.1 Introduction to `FlowControlEnv`

The `FlowControlEnv` class is a custom environment designed to simulate flow control using the Dedalus framework, interfaced with OpenAI Gym. This environment is essential for providing a controlled setting where an RL agent can learn and optimize flow control strategies.

Imports and Dependencies

```
import gymnasium as gym
from gymnasium import spaces
import numpy as np
import dedalus.public as de
import logging
from stable_baselines3 import PPO
from stable_baselines3.common.callbacks import BaseCallback
from dedalus.extras import plot_tools
import matplotlib.pyplot as plt
import os
import pickle
import time
```

These imports include necessary libraries for building the environment, handling spaces for actions and observations, logging, plotting, and integrating the RL algorithm.

Class Definition and Initialization

```
class FlowControlEnv(gym.Env):
    def __init__(self):
        super(FlowControlEnv, self).__init__()
```

The class inherits from `gym.Env`, allowing it to interface with Gym's reinforcement learning ecosystem. The `__init__` method initializes the environment.

A.2.2 Environment Setup

Grid Dimensions and Dealiasing

```
self.Nx, self.Nz = 64, 16
self.dealias = 3/2
Nx_dealias = int(self.Nx * self.dealias)
Nz_dealias = int(self.Nz * self.dealias)
```

The grid dimensions (`Nx`, `Nz`) and dealiasing factor are set to define the simulation grid's resolution. Dealiasing is used to improve the accuracy of the simulation by increasing the grid resolution.

Action Space

```
self.alpha_max = 0.03
self.action_space = spaces.Box(low=-self.alpha_max, high=self.alpha_max, shape=(1,),
dtype=np.float32)
```

The action space is defined as a continuous parameter `alpha` between `-alpha_max` and `alpha_max`. This parameter controls the agent's action within the environment, essential for adjusting flow control mechanisms.

Episode Statistics

```
self.episode_steps = 0
self.episode_count = 0
self.episode_start_time = None
```

These variables track the number of steps taken, the count of episodes, and the start time of each episode, crucial for managing the simulation's temporal aspects.

Observation Space

```
self.observation_space = spaces.Box(low=-np.inf, high=np.inf,
shape=(Nx_dealias, Nz_dealias, 2), dtype=np.float32)
```

The observation space represents the state of the environment, described by a grid with dimensions `Nx_dealias`, `Nz_dealias`, and two components. This space provides the RL agent with the necessary information to make decisions.

Dedalus Simulation Initialization

```
self.solver, self.flow, self.CFL, self.u, self.ex, self.ez, self.dist, self.coords =
self.init_dedalus_simulation()
```

This method initializes the Dedalus simulation parameters, setting up the solver, flow, CFL condition, velocity fields, and coordinates. This initialization is vital for running the physical simulations required by the environment.

A.2.3 Methods and Functionalities

Initialization of Dedalus Simulation

```
def init_dedalus_simulation(self):
    # Method to initialize Dedalus simulation parameters
    pass
```

This method sets up the Dedalus simulation by defining the problem, building the solver, and configuring initial conditions. Proper initialization ensures the physical accuracy and stability of the simulations.

Reset Method

```
def reset(self):
    # Method to reset the environment to an initial state
    pass
```

The `reset` method reinitializes the environment at the beginning of each episode, ensuring that the simulation starts from a consistent state.

Step Method

```
def step(self, action):
    # Method to take a step in the environment given an action
    pass
```

The `step` method advances the simulation by one time step based on the given action. It returns the new state, reward, a boolean indicating if the episode is done, and additional information. This method is fundamental for interacting with the RL agent.

Reward Calculation

```
def compute_reward(self):
    # Method to compute the reward based on the current state of the environment
    pass
```

The `compute_reward` method calculates the reward given to the RL agent based on the current state of the environment. This reward signals to the agent how well it is performing the task, guiding the learning process.

Rendering and Visualization

```
def render(self, mode='human'):
    # Method to render the environment for visualization
    pass
```

The `render` method provides a way to visualize the environment's state, which is helpful for debugging and understanding the agent's behavior.

A.2.4 Conclusion

The `flow_control_env.py` file provides a structured and detailed implementation of a custom flow control environment. Each function and method is designed to ensure the simulation's accuracy and to facilitate effective interaction with a reinforcement learning agent. Understanding these components is crucial for developing and optimizing RL algorithms in complex physical environments.

A.3 Conclusion

This appendix provides a detailed explanation of the project structure and key functionalities. The provided scripts and configurations form a robust pipeline for training and evaluating a PPO model in a custom flow control environment.

A.4 Results

A.4.1 Additional Figures

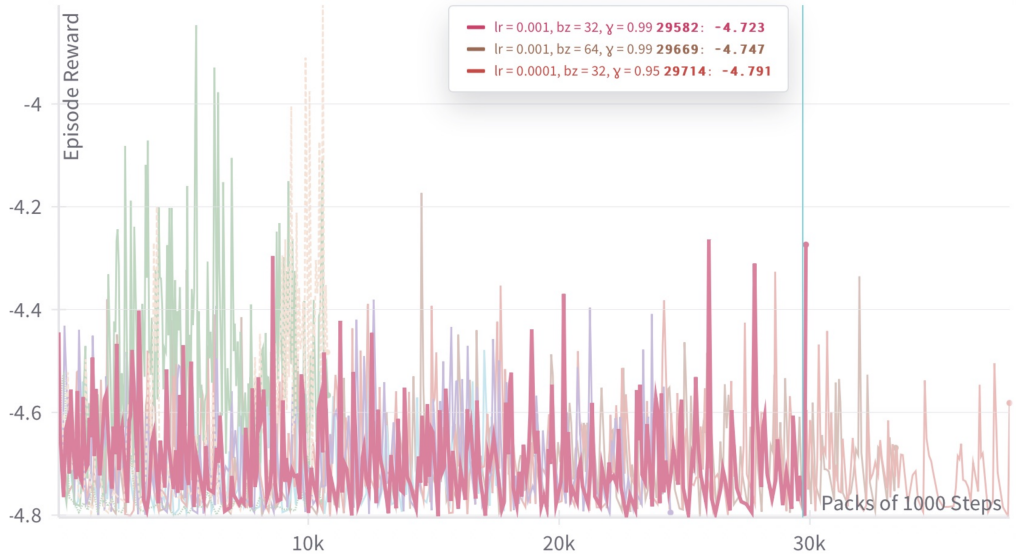


Figure A.1: Mean reward per episode (non-smoothed) for each parameter triplet. The curves appear very noisy, making it difficult to discern whether the training is improving. Nonetheless, it is noteworthy that, on average over this period, the algorithm learned very little.

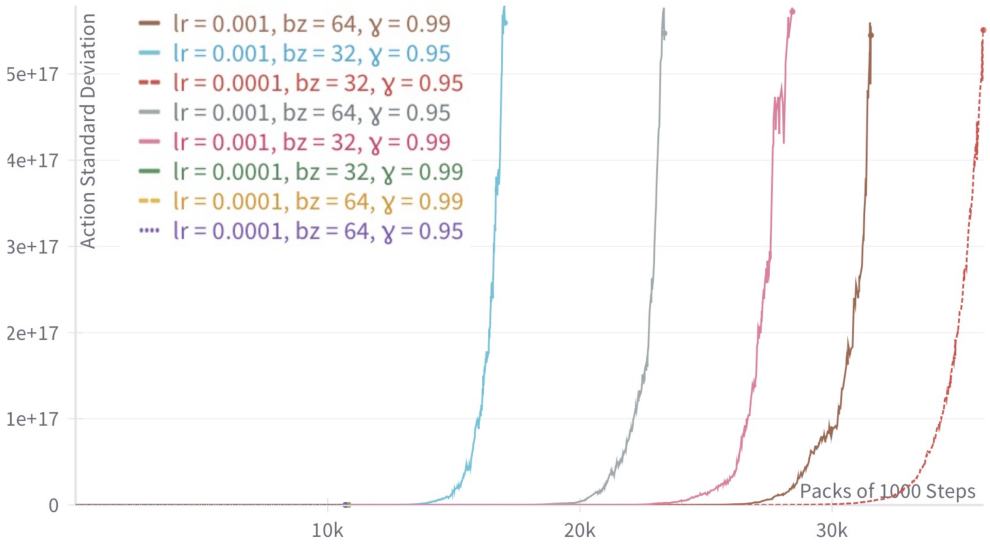


Figure A.2: Statistics of actions over a period of 1000 timesteps (standard deviation). From a certain point, there is a trend towards divergence, which is likely the cause of the program's termination.

A.4.2 Neural Network Architecture Details

Detailed Breakdown of Model Architecture

The architecture of the policy and value networks used by the PPO model is as follows:

```
Model Architecture:
ActorCriticPolicy(
  (features_extractor): FlattenExtractor(
    (flatten): Flatten(start_dim=1, end_dim=-1)
  )
  (pi_features_extractor): FlattenExtractor(
    (flatten): Flatten(start_dim=1, end_dim=-1)
  )
  (vf_features_extractor): FlattenExtractor(
    (flatten): Flatten(start_dim=1, end_dim=-1)
  )
  (mlp_extractor): MlpExtractor(
    (policy_net): Sequential(
      (0): Linear(in_features=4608, out_features=64, bias=True)
      (1): Tanh()
      (2): Linear(in_features=64, out_features=64, bias=True)
      (3): Tanh()
    )
    (value_net): Sequential(
      (0): Linear(in_features=4608, out_features=64, bias=True)
      (1): Tanh()
      (2): Linear(in_features=64, out_features=64, bias=True)
      (3): Tanh()
    )
  )
  (action_net): Linear(in_features=64, out_features=1, bias=True)
  (value_net): Linear(in_features=64, out_features=1, bias=True)
)
```

Breakdown

1. **ActorCriticPolicy**: Utilizes separate networks for policy (actor) and value estimation (critic).
2. **FlattenExtractor**: Three instances flatten the input tensor for general features, policy network features, and value network features.
3. **MlpExtractor**: Contains distinct multi-layer perceptrons (MLPs) for policy and value extraction:
   - **Policy Network (policy_net)**:
     - `Linear(in_features=4608, out_features=64, bias=True)`: Fully connected layer with 4608 input features and 64 output features, followed by a Tanh activation function.
     - `Tanh()`: Activation function.
     - `Linear(in_features=64, out_features=64, bias=True)`: Fully connected layer with 64 input features and 64 output features, followed by a Tanh activation function.
     - `Tanh()`: Activation function.
   - **Value Network (value_net)**: Identical structure to the policy network.
4. **Action_net**: Fully connected layer that maps the 64-dimensional feature space to the action space (1 dimension).
5. **Value_net**: Fully connected layer that maps the 64-dimensional feature space to the value space (1 dimension).

A.4.3 Episode Statistics

Parameters	Avg Actions/Episode	Avg Steps/Episode	Avg Duration/Episode (s)
0.001, 32, 0.95	79	46460	852.91
0.001, 64, 0.95	84	50508	814.72
0.001, 32, 0.99	81	37546	424.76
0.001, 64, 0.99	80	45999	732.76
0.0001, 32, 0.95	79	46828	795.09
0.0001, 64, 0.95	88	54797	772.71
0.0001, 32, 0.99	78	44150	712.78
0.0001, 64, 0.99	82	48016	690.83

Table A.1: Summary statistics for each parameter triplet.