Politecnico di Torino

Master of Science in Aerospace Engineering

Master's Degree Thesis

MACHINE LEARNING CONTROL OF 3D LIQUID FILM

Supervisors:
Prof. Gaetano Maria Di Cicca
Prof. Miguel Alfonso Mendez
PhD. Fabio Pino

Candidate:
Francesco Margani

Academic Year 2022/2023

Summary

The rise of machine learning has made it a crucial element in the field of fluid dynamics, and it seems very promising indeed when applied to flow control problems. This work focuses on the development of ML algorithms that can be used in the hot-dip galvanizing process to regulate the thickness of the liquid zinc film, reduce instabilities, and obtain a smoother surface, which can be critical to the performance of the final product for industrial applications. In this study, we seek to enhance the simulation of a physical problem by implementing and testing several algorithms: Bayesian Optimization (BO), Lipschitz global optimization (LIPO), and Reinforcement Learning algorithm (RL). These algorithms are applied within a simulated environment called BLEW (Boundary Layer Wiping). Our approach involves an initial analysis of the experimental setup, where we assess various configurations of jets and sensors. Additionally, we focus on optimizing the models' hyperparameters and defining appropriate actions and a cost function to improve learning performance. The results are visualized through learning curves, followed by a comparative analysis of the different algorithms employed.

Keywords: machine learning, reinforcement learning, flow control, thickness regulation, instability reduction, learning performance.

Summary

L'ascesa del machine learning ha reso fondamentale la sua integrazione con la dinamica dei fluidi tanto da diventare cruciale nei problemi di flow control. Questa tesi si concentra sullo sviluppo di algoritmi di apprendimento automatico utilizzati nel processo di zincatura a caldo per regolare lo spessore dello strato di zinco liquido per ottenere una superficie priva di difetti che potrebbero influire sulle prestazioni del prodotto finito.

Per ottenere un controllo ideale dello strato di film liquido, sono stati testati vari algoritmi, tra cui l'ottimizzazione bayesiana (BO), l'ottimizzazione di Lipschitz (LIPO) e algoritmi Reinforcement Learning(RL). Questi algoritmi vengono applicati in un ambiente simulato chiamato BLEW (Boundary LayEr Wiping). Inizialmente, è stata eseguita un'analisi preliminare per determinare il corretto compromesso tra posizione dei getti e quella dei sensori. Successivamente, sono stati ottimizzati gli iperparametri dell'algoritmo di machine learning e sono state affinate la funzione di costo e le azioni dei getti. Infine, sono state eseguite simulazioni su un elevato numero di episodi per osservare l'apprendimento dell'agente, valutato attraverso le curve di apprendimento e il test del modello allenato su un episodio di valutazione.

Acknowledgements

This Master's Thesis project aligns with a wider research area pursued at the Von Karman Institute (VKI) by my advisor, F. Pino, and my supervisor, Prof. M. A. Mendez. I express my heartfelt gratitude to both of them for their invaluable assistance and unwavering guidance throughout the course of this Master's thesis. Their exceptional creativity and profound ideas have left an indelible mark on my mind.

Additionally, I would like to extend my sincere appreciation to my supervisor at Politecnico di Torino, Prof. Gaetano Maria Di Cicca. It is his support that has made this extraordinary experience at VKI possible.

Table of Contents

List of Tables

List of Figures

Acronyms

VKI Von Karman Institute

BLEW Boundary LayEr Wiping

NS Navier-Stokes

PDE Partial Different Equation

FV Finite Volume

ML Machine Learning

RL Reinforcement Learning

BO Bayesian Optimization

GPr Gaussian Process regression

LIPO Lipschitz Global optimization

PPO2 Proximal Policy Optimization

DDPG Deep Deterministic Policy Gradient

MLP Multi Layer Perception

LC Learning Curves

HPO Huper Paramteres optimization

LHS Latin Hypercube Sampling

ANN	Artificial Neural Network
EMA	Exponential Moving Avarage
Std	Standard Deviation
env	Environment
obs	Observation point

List of Symbols

$[h]$ dimensionless thickness of the liquid film
$[x]$ streamwise reference length
U_p vertical velocity of the metal strip
τ_g shear stress distribution
p_g shear stress distribution
C_a capillary number
Re Reynolds Number
ν_l kinematic viscosity liquid
μ_l dynamic viscosity liquid
ρ_l liquid density
σ_l surface tension liquid
g gravity constant
d jet diameter
Z stand-off distance
ΔP_N stagnation pressure inside the nozzle
$\hat{q}$ flow rate
$s(t)$ state
$a(t)$ control action
$R(s(t), a(t))$ Reward
$\pi(s(t), t)$ policy
γ discount factor
v_π value of state s
Q_π Q-value
M Mach number
h enthalpy
U_j exit flow velocity
Π_g wiping number
ρ_g gas density
p_s pressure at impinging point
τ_{0m} maximum shear stress

Chapter 1

Introduction

1.1 Physical overview

One of the great challenges in metallurgy is the optimization of the process of galvanization, which is a crucial part of the proprieties of the final product. Using a coat of zinc metal alloy as a protection, the metal can be protected from corrosion and oxidation improving mechanical performance.

The galvanization industry has a worldwide production capacity of around 100 million tons of steel per year across over 450 galvanizing lines; there have been advancements in the industry, such as the introduction of fresh chemical coating compositions, novel steel grades, and alternative surface treatments. Developing these innovative products necessitates changes to the process to attain the desired final coating quality.

One of the most useful types of galvanizing in siderurgy is continuously hot dip coating, in which a metal strip is immersed in a molten liquid zinc bath and then moved vertically to reduce and regulate the thickness with some nozzles. It is done by some rolls that give the metal strip a certain speed, and after the liquid bath, some high-speed gas jet can control the thickness of the zinc, leaving a thinner and uniform coat in the substrate. This kind of application is called jet wiping.

But after an initial phase of gas jet wiping, some instabilities may occur: they resemble waves and are called undulation, and may cause problems in the post-processing of the final product and can reduce its mechanical features. The goal is to reduce these imperfections as much as possible, and the challenge is to use a new system of Machine Learning algorithms to control these instabilities.

This is a closed-loop control, which is a system with mechanical or electronic devices that automatically try to maintain a desired state or set point without human interaction. In this case study, it is made with a series of sensors that take as an input the liquid film's thickness and a controller with an ML algorithm, which dictates the actions for the nozzles.

For this project, the sensors that can be chosen should be laser triangulation or video inspector, while the actuators should be hydrodynamic actuators or magnetic actuators. By the way, the facility should be built in Canada in a few years as shown in Figure 1.1.

Continuous research is also conducted to enhance the process's dependability and durability and to be competitive in cost and time.

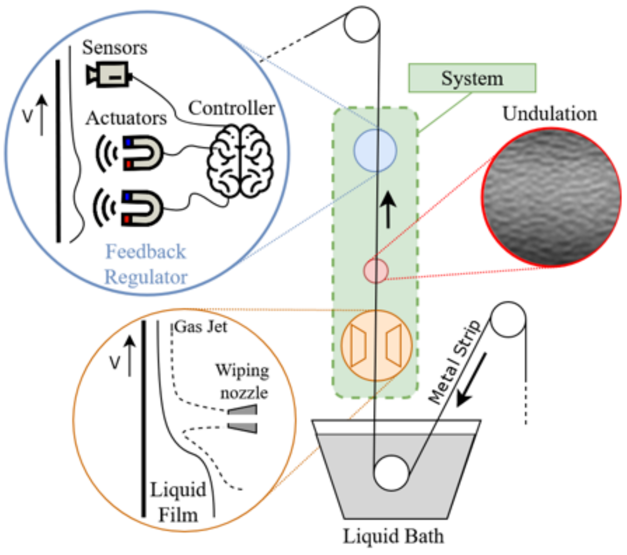


Figure 1.1: *Hot-dip galvanization process with controller for undulation. Schematic of the galvanization process with two phases: a normal jet wiping and an undulation control made with the system of actuators, controller, and sensors [1].*

1.2 From Navier-Stokes equation to BLEW

The thesis focuses on the dynamics of a liquid film that falls along a vertical plate moving upwards, while a gas jet is directed onto it. It is assumed that the gas jet is not affected by the presence of the liquid.

Starting with the Navier-Stokes equations, the focus is on analyzing the flow behavior of the liquid film[2].

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{1.1}$$

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{v} + \mathbf{f_v} \tag{1.2}$$

All of the mathematical models studied are based on the Navier-Stokes equation and its associated boundary conditions as a 'long-wave' formulation. This formulation is obtained by proportionally reducing the dimensions perpendicular to the flow direction using a reference length $[h]$ which is the dimensionless thickness of the liquid film and it is significantly smaller than the reference length in the flow direction $[x]$ oriented in the direction of gravity.

$$[h] << [x] \tag{1.3}$$

Before starting to liquid film fluid dynamics description as shown in Figure 1.2, is important to define the references quantities that are used to scale the problem in Table 1.1

Table 1.1: reference quantities used for the Navier-Stokes long-wave formulation [2]

Reference Quantity	Definition	Expression
$[h]$	$(\nu_l[u]/g)^{1/2}$	$(\nu_l U_p/g)^{1/2}$
$[x]$	$[h]/\varepsilon$	$(\nu_l U_p/g)^{1/2}\,\mathrm{Ca}^{-1/3}$
$[u]$	U_p	U_p
$[v]$	εU_p	$U_p \mathrm{Ca}^{1/3}$
$[p]$	$\rho_l g[x]$	$(\mu_l \rho_l g U_p)^{1/2}\,\mathrm{Ca}^{-1/3}$
$[\tau]$	$\mu_l[u]/[h]$	$(\mu_l \rho g U_p)^{1/2}$
$[t]$	$[x]/[u]$	$(\nu_l/U_p g)^{1/2}\,\mathrm{Ca}^{-1/3}$

In the long-wave formulation of the problem, the dimensionless continuity and the momentum equation in the x and the y directions reduce to the boundary layer equations, which are written considering dimensionless variables scaled concerning the quantities in Table 1.1 are indicated with a hat (ex. $\hat{h} = h/[h]$) as done in [2]:

$$\begin{aligned} \partial_{\hat{x}}\hat{u} + \partial_{\hat{y}}\hat{v} &= 0, \\ \varepsilon\,\mathrm{Re}\left(\partial_{\hat{t}}\hat{u} + \hat{u}\partial_{\hat{x}}\hat{u} + \hat{v}\partial_{\hat{y}}\hat{u}\right) &= -\partial_{\hat{x}}\hat{p}_l + \partial_{\hat{y}\hat{y}}\hat{u} + 1, \\ 0 &= \partial_{\hat{y}}\hat{p}_l, \end{aligned} \tag{1.4}$$

where $\hat{p}_l$ is the pressure in the liquid, $\hat{u}$ and $\hat{v}$ are the streamwise and cross-stream velocity components, $\varepsilon = [h]/[x] = Ca^{1/3}$ is the film parameter, with $Ca = \mu_l U_p/\sigma$ the capillary number and $Re = [u][h]/\nu_l = \left(U_p^3/g\nu_l\right)^{1/2}$ is the global Reynolds number of the process. The formulation proposed in this work is also valid for the experimental conditions encountered at the von Karman Institute (VKI), operating with water ($\mu_l = 0.001 Pa*s$, $\sigma = 0.07 N/m$ at $U_p = 1m/s$ (i.e. $Ca = 0.003 - 0.03$).

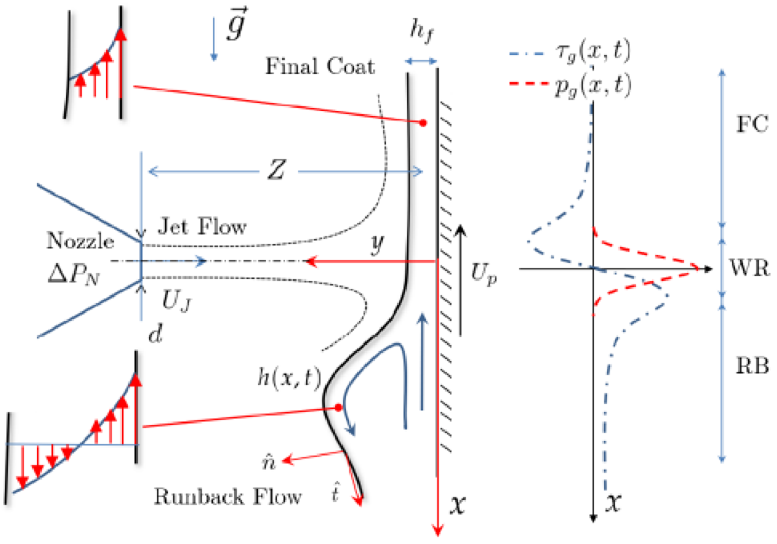


Figure 1.2: *Schematic jet wiping*
A nozzle with an opening d and a stagnation pressure ΔP_N releases a jet flow at a distance Z from a dip-coated substrate moving at a speed U_p. Thickness h_f is modelled by pressure p_g and shear stress distribution τ_g[2]

To reduce the complexity of the system is possible to use the Integral Boundary Layer (IBL) model to represent this as a 1D problem (more details in [2]). After that, a numerical method can be used to solve this PDEs hyperbolic equations that can be written in the general form as a model equation:

$$\partial_t \mathbf{V}(x,t) + \partial_x \mathbf{F}(x,\mathbf{V}) = \mathbf{S}(x,t,\mathbf{V}) \tag{1.5}$$

with $\mathbf{V}$ the state vector of the problem, $\mathbf{F}$ the conservative flux and $\mathbf{S}$ the source term.

The state vector is represented by:

$$\mathbf{V} = \begin{bmatrix} \hat{h} \\ \hat{q} \end{bmatrix} \tag{1.6}$$

where

$$\vec{U} = \left(\hat{h}, \hat{q}_x, \hat{q}_z\right)^{\mathrm{T}} \quad \hat{q}_z = \int_0^{\hat{h}} \hat{w} d\hat{y} \quad \hat{q}_x = \int_0^{\hat{h}} \hat{u} d\hat{y} \tag{1.7}$$

so it is represented by the value of the dimensionless high of liquid film $\hat{h}$ and the two flow rates $\hat{q}_z$ and $\hat{q}_x$.

The flux term is:

$$\mathbf{F} = \begin{bmatrix} -\hat{q} \\ -\hat{\mathcal{F}}(\hat{h}, \hat{q}) \end{bmatrix} \tag{1.8}$$

where $\hat{q}$ is the volumetric flow rate and $\mathcal{F}$ is the advection term that is in this case

$$\mathcal{F} = \int_0^{\hat{h}} \hat{u}^2 d\hat{y} \tag{1.9}$$

The source term is

$$\mathbf{S} = \begin{bmatrix} 0 \\ -\hat{h}\partial_{\hat{x}}\hat{p}_g + \hat{h} + \frac{\epsilon^3}{Ca}\hat{h}\partial_{\hat{x}\hat{x}\hat{x}}\hat{h} + \hat{\tau}_g + \hat{\tau}_w(\hat{q}, \hat{h}) \end{bmatrix} \tag{1.10}$$

It includes the pressure gradient, surface tension, gravity, and shear stress effect. In this case, the IBL has been used, and for this, the β parameter is set to 1 as done in [2] equation 6.2.

Two methods can be used to solve this set of PDEs: methods based on the Riemann problem solution or methods based on the Lax-Friedrics Lax-Wendroff scheme. The first one is more useful in case of shock, while the second has less computational cost. Since this problem has no shock wave, the second one can be the best option. BLEW (Boundary LayEr Wiping) code is based on the solution of the model equation using the finite volume method (FV) that in its standard form for this problem can be written as:

$$\mathbf{V}_i^{k+1} = \mathbf{V}_i^k - \frac{\Delta t}{\Delta x}\left[\mathbf{F}^+ - \mathbf{F}^-\right] + \Delta t \mathbf{S}_i^k \tag{1.11}$$

that allows us to solve the system of PDE with numerical simulations as done in [2]. In this case, the BLEW code is developed utilizing the preceding equations to operate alongside machine learning algorithms, whereas the fluctuation arises from authentic perturbations whose values are derived from Large Eddy Simulation (LES).

Thus, this work aims to edit the code already implemented at VKI using machine learning methods to control a complex 3D instability of a falling liquid film over a moving substrate.

Indeed, using these types of algorithms, it is possible to analyze large amounts of data and identify patterns that human operators may overlook. By training models on historical data, machine learning can accurately predict liquid thickness based on input variables such as flow rate, temperature, viscosity, and other relevant factors and also can lead to improved efficiency, reduced material waste, and enhanced overall process performance.

Chapter 2

Machine Learning (ML)

In closed-loop control, several ML algorithms can be used. This thesis aims to focus on two types of Reinforcement Learning algorithms and Bayesian Optimization. These machine learning control methods differ in the way they are learned and modify the controller parametrization. In this project, machine learning algorithms are implemented in Python using respectively the Skopt [3] library for the Bayesian Optimization and Lipschitz global Optimization (LIPO) while Stable Baselines [4] library (based on TensorFlow v.1.14 [5]) has been used for RL algorithm.

2.1 Optimal control and Reinforcement Learning (RL)

In control problem [6] we are dealing with a dynamical system that evolves according to some function. Some calls this the plant others call it the environment. The goal is to keep the system as close as possible to a certain trajectory in the system's phase space. To do that, there is a controller (or for others an agent) that acts on the system, and we have a performance measure that tells us how good or bad we are doing. We can craft this as a reward to maximize or a cost to minimize, but either way, our goal is to find the optimal set of actions. The reward or cost is computed within a specific time range, which we will call an episode, within which our controller/agent interacts with the plant/environment.

The environment is defined as

$$\dot{s(t)} = f(s(t), a(t)) \tag{2.1}$$

$$s(0) = s_0 \tag{2.2}$$

where $s(t)$ is the state and $a(t)$ is the control action.

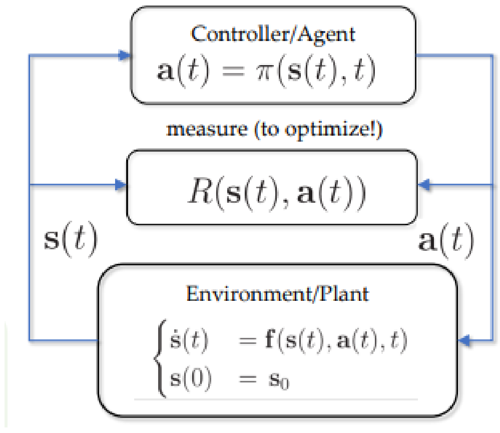

Controller/Agent
$\mathbf{a}(t) = \pi(\mathbf{s}(t), t)$
measure (to optimize!)
$R(\mathbf{s}(t), \mathbf{a}(t))$
$\mathbf{s}(t)$
$\mathbf{a}(t)$
Environment/Plant
$\begin{cases} \dot{\mathbf{s}}(t) &= \mathbf{f}(\mathbf{s}(t), \mathbf{a}(t), t) \\ \mathbf{s}(0) &= \mathbf{s}_0 \end{cases}$

Figure 2.1: RL scheme
The learning subject of reinforcement learning is called the agent. The state in the environment is perceived by the agent, and the state at step t is denoted by s(t). The agent chooses an action from the set of actions available at the current state. The action taken at step t is denoted by a(t)

We have a variational problem: we must find a function that optimizes another function.

This function from states to action is what some of us call control law others call policy and is represented by $\pi(s(t), t)$ that decides the action $a(t)$ at each time step.

In Figure 2.1, is possible to see the RL scheme with all the elements that characterize it and their interactions. It is clear to see that the action influences the state and the reward in an iterative process aimed at reducing/maximizing the cost/reward R.

In RL, the value function represents the reward that an agent receives for being in a particular state. To determine this reward, the agent's actions must be taken into consideration, which is where policy comes in. The Bellman equation is a fundamental formula that calculates the expected value of rewards received from a given policy, taking into account the discount factor γ that balances the importance of immediate and future rewards. It has a value between 0 and 1: a value of 0 means that more importance is given to the immediate reward and a value of 1 means that more importance is given to future rewards

Bellman equation:

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} ... [S_t = s] \tag{2.3}$$

The Q-Value, also known as Quality Value, is a similar algorithm to the Bellman equation that estimates the optimal state-action values. It represents the value of taking a particular action in a given state with a policy.

$$Q_{\pi}(s,a) = E_{\pi}[G_n|s_n = s, a_n = a] \tag{2.4}$$

Once the optimal Q-Values have been determined, the optimal policy can be easily defined as the action with the highest Q-Value for each state.

$$\pi^{*}(s) = argmaxQ^{*}(s,a) \tag{2.5}$$

Q-Learning is a popular technique that involves observing an agent playing a task and improving its estimates of the Q-Values over time. Once the Q-Values are accurate enough, the optimal policy can be achieved by choosing the action with the highest Q-Value

In our case study, the environment is the BLEW env, represented by a fluid film with some waves to simulate wave motion. It consists of a system of observation points and nozzles that receive as input the thickness of the fluid film and as an output of the controller the action for the nozzles. To evaluate how good the selected actions are, a cost function gives a reward for each episode, and the goal of machine learning is to adjust this action to maximize this reward. The policy can change its weights according to the reward so that after a sufficient number of episodes, it should learn what to do in that environment for each perturbation. Several algorithms can be used to optimize this process, but the most favorable in terms of computational cost and efficiency are Bayesian Optimization(BO), Proximal Policy Optimization (PPO2), and Deep Deterministic Policy Gradient (DDPG).

2.2 Bayesian Optimization (BO)

Bayesian optimization (BO) is a method used for optimizing a function by creating a probabilistic model of it, typically using a Gaussian Process regression (GPr) model. It has been implemented and analyzed by Shahriari [7].

It has two important parts: the first is a model that uses a probabilistic surrogate model to guess how an unknown objective function works, based on assumptions. It also includes a way to describe how data is generated. The second part is a loss function to measure how good a sequence of queries is. This is often done using regret, which can be either simple or cumulative. Mathematically, BO consists in finding a global maximum or minimum of an objective function f:

$$\mathbf{x}^{\star} = \arg\max_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \tag{2.6}$$

As it is shown in Figure 2.2, the goal in this case for a 1D function is to minimize the distance between the black and the dashed line, and we can see how the acquisition function works: it is low in the sampled point, where the value is known, while it is high in the exploration phase; Moving on step by step $(n = 2, n = 3, n = 4)$, it tries to rebuild the original function.

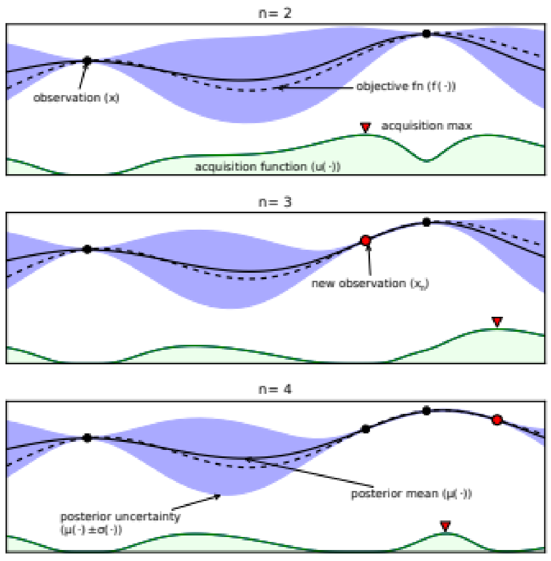
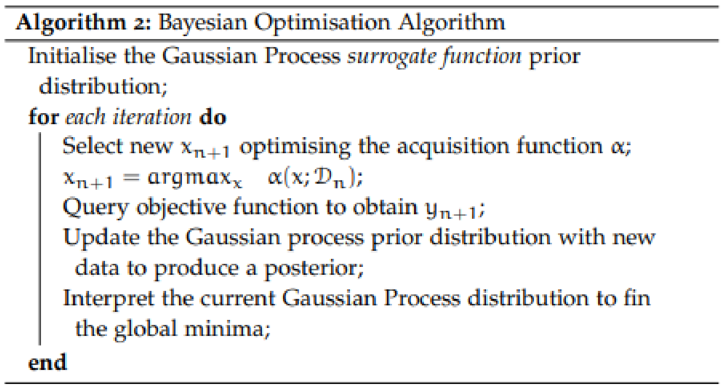


Figure 2.2: *Bayesian Optimization*
The visual representations depict the average value and confidence intervals, which are determined using a probabilistic model that represents the objective function in 3 steps. Although the actual objective function is depicted in the illustrations, it remains unknown in practical scenarios.[7].

The GPr model estimates the probability of a particular reward given the observations (input-output pairs) of the function being optimized. In BO, this is usually the cumulative reward function $R(w)$.

To build the GPr model, a set of tested weights W and their associated cumulative rewards R are used. The GPr model computes the probability of a reward for a new set of weights W. This is done by assuming that the new candidate solutions belong to the same GPr as the observed data (W, R) and using a kernel function to compute the covariance matrix.

BO combines the GPr model with a function that suggests where to sample next. Different variants of BO exist, each providing its own exploration/exploitation balance. BO is a popular black-box optimization method for expensive cost functions.



```
Algorithm 2: Bayesian Optimisation Algorithm
Initialise the Gaussian Process surrogate function prior
  distribution;
for each iteration do
    Select new $x_{n+1}$ optimising the acquisition function $\alpha$;
    $x_{n+1} = \operatorname{argmax}_x \quad \alpha(x; \mathcal{D}_n)$;
    Query objective function to obtain $y_{n+1}$;
    Update the Gaussian process prior distribution with new
      data to produce a posterior;
    Interpret the current Gaussian Process distribution to fin
      the global minima;
end
```

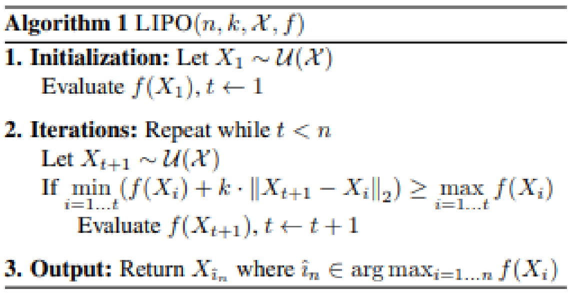
Figure 2.3: *BO pseudo-code*
The essence of this algorithm lies in the process of adjusting an initial, or prior, belief based on newly sampled information, subsequently yielding a final, or posterior, estimation..[7]

2.3 Lipschitz Global Optimization (LIPO)

The primary concept of the Lipschitz Global Optimization(LIPO) as presented in [8] involves the maintenance of a piecewise linear upper bound for the function $f(x)$. This method is utilized to determine which values of x to assess during the optimization process. Once certain values of x, such as x_1, x_2, ..., and x_t, have been evaluated, a basic upper bound for $f(x)$ can be created as follows:

$$U(x) = min(f(x_i) + k||x - x_i||^2) \tag{2.7}$$

The Lipschitz constant for $f(x)$ is represented by k. A straightforward technique known as LIPO involves selecting random points and determining if the upper bound for the new point surpasses the best point seen so far. If it does, the algorithm selects it as the following point to evaluate as in the example for a 1D function in Figure 2.5. In the end, it tries to rebuild the original function.



```
Algorithm 1 LIPO$(n, k, \mathcal{X}, f)$
1. Initialization: Let $X_1 \sim \mathcal{U}(\mathcal{X})$
    Evaluate $f(X_1)$, $t \leftarrow 1$
2. Iterations: Repeat while $t < n$
    Let $X_{t+1} \sim \mathcal{U}(\mathcal{X})$
    If $\min_{i=1\ldots t}(f(X_i) + k \cdot \|X_{t+1} - X_i\|_2) \geq \max_{i=1\ldots t} f(X_i)$
        Evaluate $f(X_{t+1})$, $t \leftarrow t + 1$
3. Output: Return $X_{\hat{\imath}_n}$ where $\hat{\imath}_n \in \arg\max_{i=1\ldots n} f(X_i)$
```

Figure 2.4: *LIPO pseudo-code*
The inputs of the LIPO algorithm are a number n of function evaluations, a Lipschitz constant k, the input space X and the unknown function f. At each iteration, a random variable is sampled uniformly over the input space X and the algorithm decides whether or not to evaluate the function at this point [8].

2.4 Proximal Policy Optimization (PPO2)

PPO2[9] propose a novel set of policy gradient techniques for reinforcement learning that involve iterative sampling of data through interaction with the environment, followed by optimization of a "surrogate" objective function using stochastic gradient ascent.

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta \left(a_t \mid s_t\right) \hat{A}_t\right] \tag{2.8}$$

where π_θ is the policy and A is the estimator of advantage.

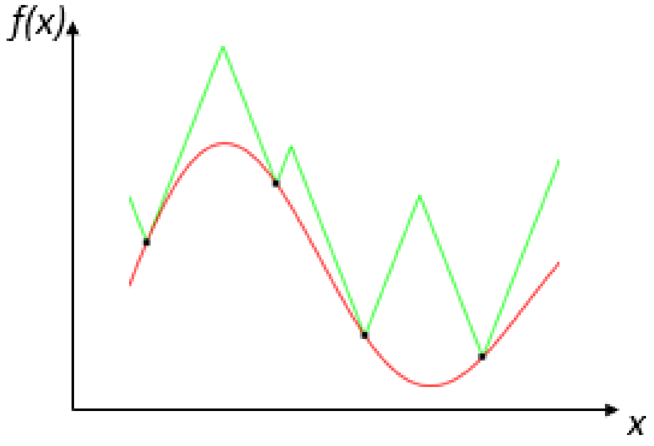
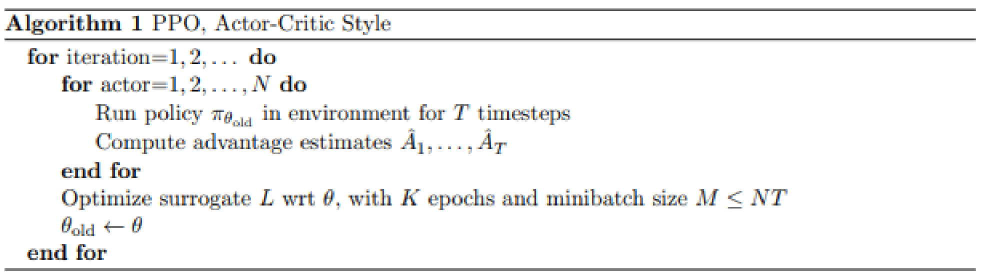


Figure 2.5: *LIPO on 1D function*
This figure shows a plot of a simple f(x) in red with a plot of its associated upper bound U(x) in green [8]

The main idea is that after an update, the new policy should be not too far from the old policy. For that, PPO uses clipping to avoid too large an update.

In contrast to conventional policy gradient methods, which perform a single gradient update per data sample, we introduce a new objective function that facilitates the use of multiple epochs of minibatch updates. This approach, known as proximal policy optimization (PPO), shares some advantages of Trust Region Policy Optimization (TRPO), an iterative procedure for optimizing policies, with guaranteed monotonic improvement. However, based on our empirical results, it is easier to implement, more flexible, and has superior sample complexity.



```
Algorithm 1 PPO, Actor-Critic Style
for iteration=$1, 2, \ldots$ do
    for actor=$1, 2, \ldots, N$ do
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for
```

Figure 2.6: *PPO2 pseudo-code*
Each iteration, each of N (parallel) actors collect T timesteps of data. Then we construct the surrogate loss on these NT timesteps of data, and optimize it [9]

2.5 Deep Deterministic Policy Gradient (DDPG)

DDPG [10] is another kind of black box RL deterministic algorithm based on an actor-critic model that can operate in a continuous action space. Two regressors are needed to evaluate an actor-critic model each timestep based on a neural network to evaluate the Q-function and to learn the policy (actor) and the oracle (critic).

In the Actor-Critic method, the policy is referred to as the actor that proposes a set of possible actions given a state, and the estimated value function is referred to as the critic, which evaluates actions taken by the actor based on the given policy. The critic $Q(s, a)$ is learned using the Bellman equation as in Q-learning. In other words, DDPG uses data and the Bellman equation to learn the Q-function and uses the Q-function to learn the policy. The objective is to evaluate the gradient of the two functions and update both of them. The DDPG policy is an MLP (Multi-Layer Perception) policy, which is a neural network that contains more than one layer; each one is composed of neurons and information passes from layer to layer, from input to output, as in figure 2.7 where is possible to see the 2 regressors and how they work with the neural network. We present the neural network structure utilized in this study: the diagram illustrates the interconnection between the Π network and the Q network. Within this framework, the intermediate layers facilitate the mapping of the present state and the action (generated by the Π network) to the central component of the Q network.

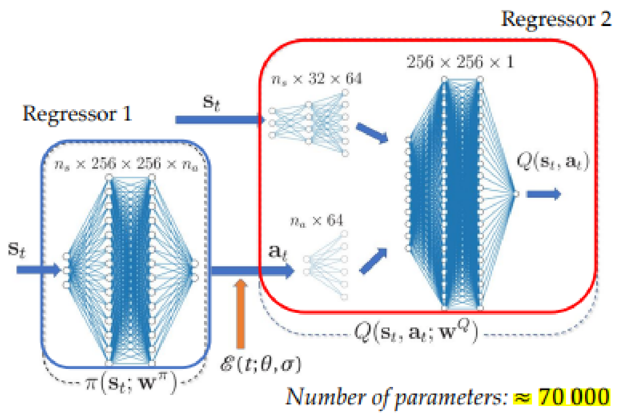
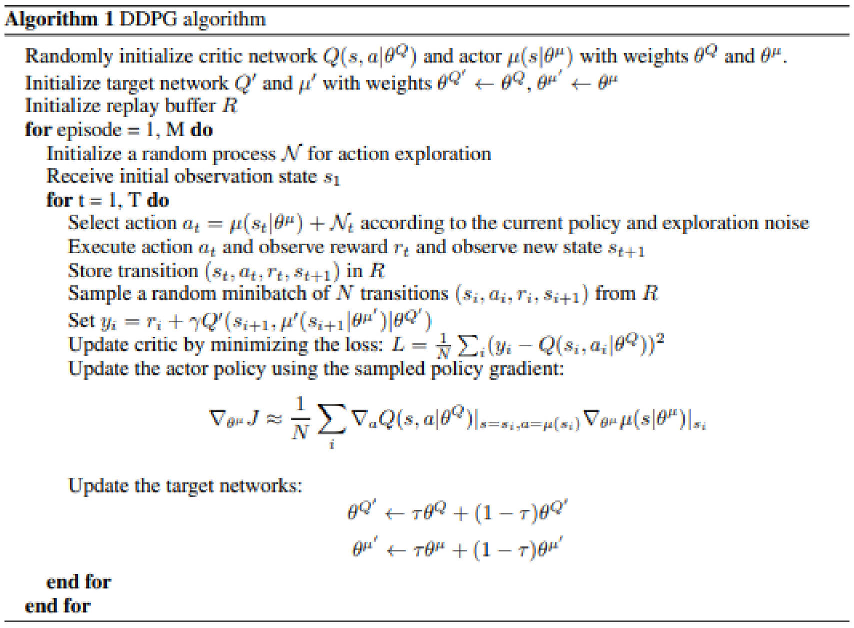

Regressor 2
$256 \times 256 \times 1$
$n_s \times 32 \times 64$
Regressor 1
$\mathbf{s}_t$
$n_s \times 256 \times 256 \times n_a$
$Q(\mathbf{s}_t, \mathbf{a}_t)$
$n_a \times 64$
$\mathbf{s}_t$
$\mathbf{a}_t$
$Q(\mathbf{s}_t, \mathbf{a}_t; \mathbf{w}^Q)$
$\mathscr{E}(t; \theta, \sigma)$
$\pi(\mathbf{s}_t; \mathbf{w}^\pi)$
Number of parameters: ≈ 70 000

Figure 2.7: *ANN for DDPG, actor-critic model*
Throughout the exploration stage, the two networks are effectively disconnected due to the inclusion of the stochastic element ϵ, which facilitates the exploration of the space of possible actions [6]



```
Algorithm 1 DDPG algorithm
  Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
  Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
  Initialize replay buffer $R$
  for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
      Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
      Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
      Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
      Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
      Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
      Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
      Update the actor policy using the sampled policy gradient:
          $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
      Update the target networks:
          $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
          $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    end for
  end for
```

Figure 2.8: *DDPG Pseudo-code*
Starting from random values, in each iteration, both actor and critic are updated by minimizing loss [10]

2.6 OpenAI Gym Tutorials

Using an ML algorithm to work on BLEW can be useful use to solve some trivial problems. For example, OpenAI, the largest artificial intelligence organization, provides a Gym [11] with some simple environments that can be used to better understand how the ML algorithm works and to prepare users for more complex environments. More information can be found in the Gym documentation and on GitHub[12].

It can be interesting to use some RL algorithms like PPO2 and DDPG to solve some trivial problems and see how the algorithm's policy can learn.

- Mountain Car

  This simple environment consists of a car that is initially in a valley. The goal is to give the car actions to reach the mountain, and only in this case it receives a positive reward; otherwise, it receives a negative reward. If the actions cannot give the car enough power, the reward is $-0.1 * Actions^1$, but if the mountain car reaches the goal, then a positive reward of +100 is added to the negative reward for each timestep. To solve this environment a DDPG has been used, and after a 500000 timestep of training the policy can give the action to reach the valley to the car as shown in figure 2.9.

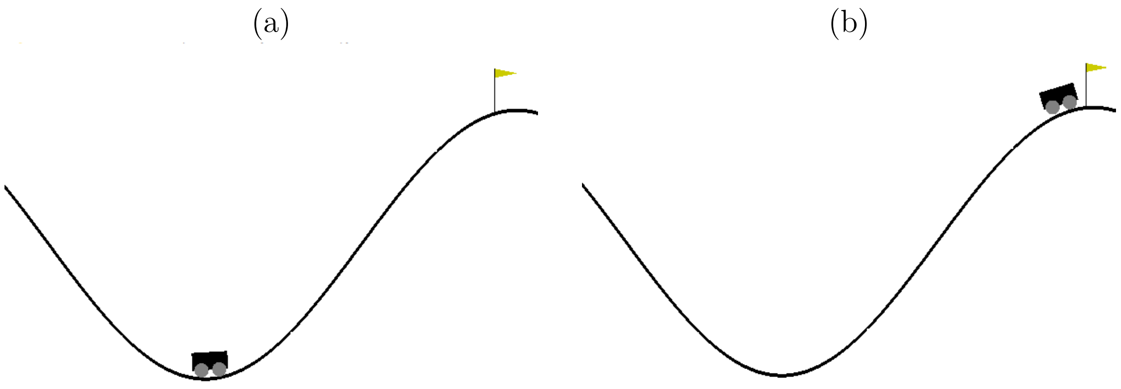


Figure 2.9: *Mountain car*
In the first figure (a), the car is in the valley. In the second (b), after the training of the agent, it reaches the mountain

It may also be interesting to observe the ability to learn from the policy for this environment. In ML it may be useful to plot learning curves representing the value of the reward for each episode. They start with a random value, and if ML learns well, the reward should increase at each step until it reaches its maximum. Visualizing this in Tensorboard, a Tensorflow dashboard, and then seeing the LC as in Figure 2.10 is possible.

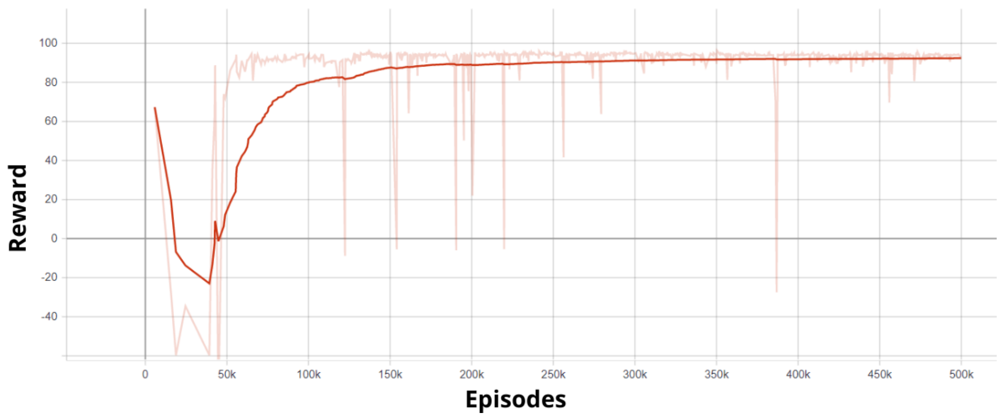


Figure 2.10: *LC Mountain Car*
The reward starts from a random point, and after many iterations, it converges to the expected value

- Lunar Lander

  This is another trivial problem of OpenAI gym. In this case, the goal is to give action to the rover to allow it to land between the two flags. In this game, the lander will receive negative points if it moves away from the landing pad. Similarly, if it crashes, the lander will lose an extra 100 points, while coming to a complete stop will earn it an additional 100 points. Each leg of the lander that has contact with the ground is worth 10 points. So the only way to earn a positive reward is to land between the two flags as shown in Figure 2.11 To solve this environment a PPO2 has been used with about 500000 episodes of training

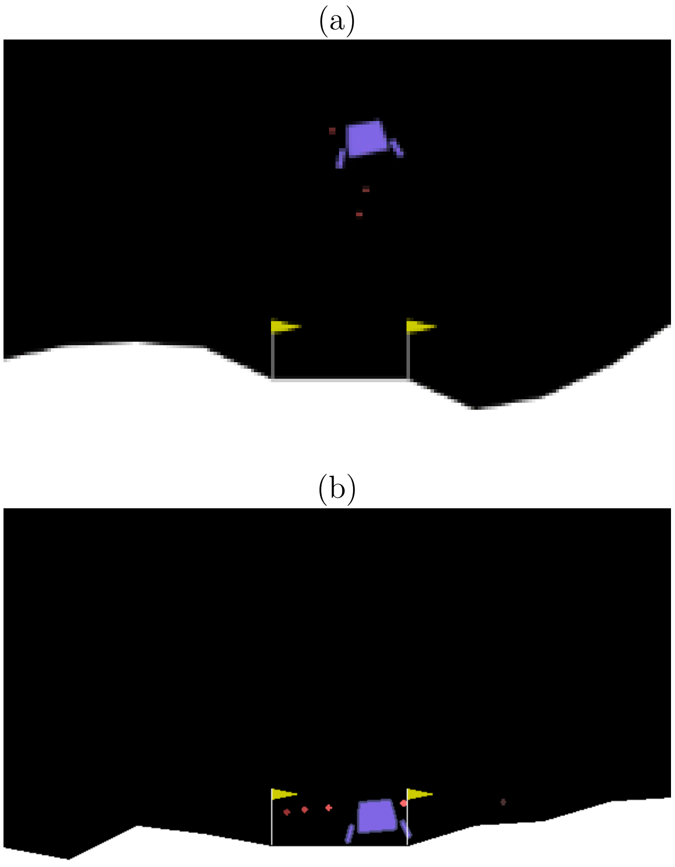


Figure 2.11: *Lunar Lander*
In the first figure (a) the rover is landing. Without the agent's learning, it will crash at a random point. After the training, it lands between the two flags (b)

As for Mountain cars, it can be useful to see the learning curves for this environment with Tensorboard in Figure 2.12

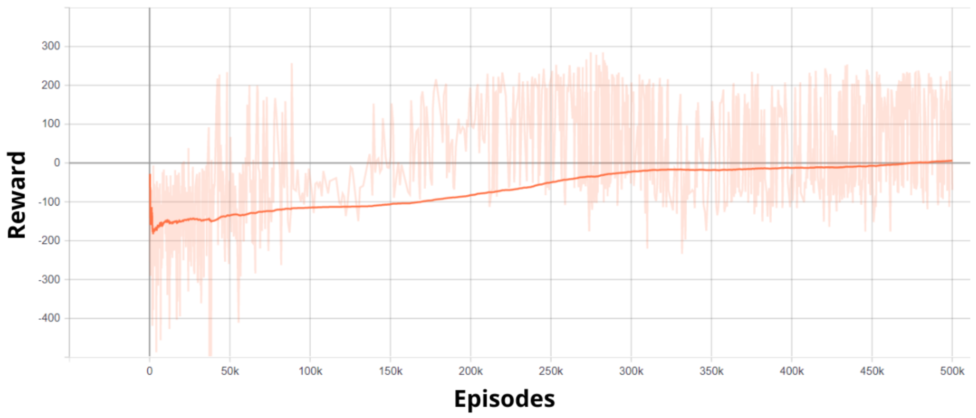


Figure 2.12: *LC Lunar Lander*
After a certain number of episodes, the reward converges as we expected

Chapter 3

Jet Wiping

The goal of finding the best parameters that can optimize the height of the fluid film, h, is the reward for reinforcement learning. Thus, the most important parameters that can be analyzed are the characteristics of the nozzle interacting with the flow, considering a decoupling between the airflow and the film flow to obtain better results for the numerical simulation so that the jet is not affected by the fluid as it impinges on the flat plate. The first approach is to set the parameters such as the distance H and the nozzle diameter d. It is also interesting to determine the exit velocity max U_j taking into account the throttling condition of the nozzle and thus the value of the pressure in the nozzle P_j. Starting from this, let's compute the shear stress and pressure distribution to model the liquid film thickness, and it is the problem set-up as shown in Figure 3.1.

3.1 Jet characterisation

Impinging circular jets are a special type of jet that is commonly used in jet wiping and characterized by their circular shape and perpendicular impact on the surface to be coated as described by Beltaos e Rajaratnam [13]. Considering a jet impinging normally on a smooth surface, such as the metal strip, it is possible to distinguish three distinct areas of the flow as shown in Figure 3.2. The first region has the same characteristics as a free jet with a potential core, where $V_j = V_{0j}$, the transitional zone, and the self-similar region. In the second region, the flow has a deflection caused by the metal sheet, it is no longer perpendicular to the surface but parallel. Right after impingement, the flow re-accelerates, so a peak of velocity and pressure fluctuations arise, with a generation of turbulence close to the wall. In the end, there is the wall jet region, where the flow is aligned with the foil.

The area where the flow directly hits the boundary is known as the impingement region, and it is where the most intense hydrodynamic forces are experienced. In

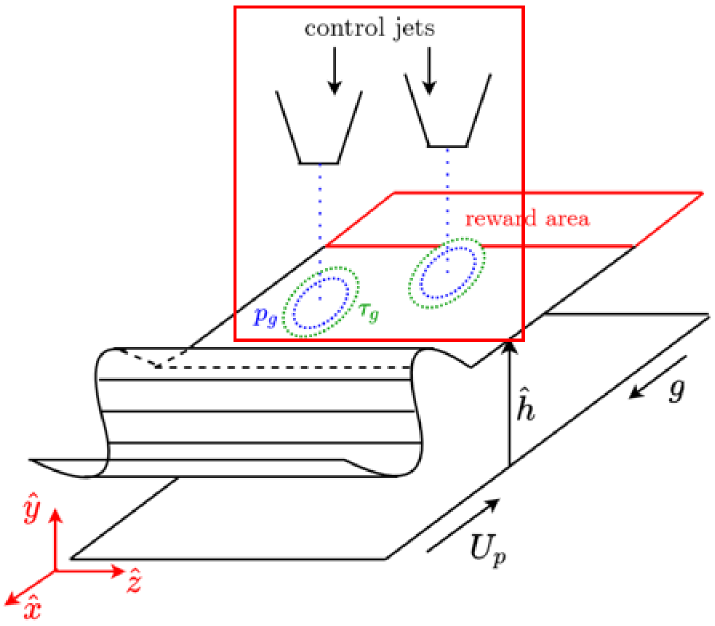


Figure 3.1: *Control problem set-up*
This summarizes the goal of this section, which is to model the jets and its related p_g and τ_g to give actions for reducing liquid film instabilities [1].

this region, both the wall shear stress and pressure gradients are substantial, so the study of them is necessary to analyze the interaction between jet wiping and liquid film to reduce its thickness.

3.2 Exit flow velocity

It should be interesting to evaluate the velocity flow limit: it is strictly connected to the number of Mach because the goal is to avoid the choking condition in the nozzle.

In this case study, it is necessary to use a compressible formulation of Bernoulli for a gas (negligible gravitational effects).

$$h_2 + \frac{U_2^2}{2} = h_1 + \frac{U_1^2}{2} + W - Q + W_v \tag{3.1}$$

where U is the flow velocity, h is the enthalpy, W is shaft work that could be introduced by a machine, Q is the exchanged heat and W_v is the energy lost due to friction, rotationality, and any other effect. In this case point 1 is the nozzle chamber and 2 is the outlet. So, in 2 it can be possible to use the value of T_{amb} e

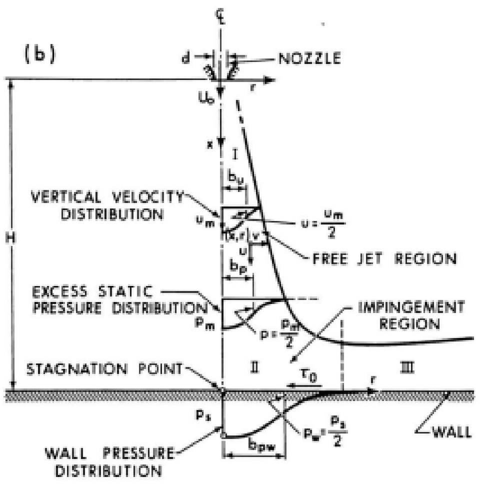

(b)
NOZZLE
VERTICAL VELOCITY DISTRIBUTION
FREE JET REGION
EXCESS STATIC PRESSURE DISTRIBUTION
IMPINGEMENT REGION
STAGNATION POINT
WALL
WALL PRESSURE DISTRIBUTION
H
I
II
III

Figure 3.2: *Jet flow region*
Considering a jet impinging normally on a plane smooth surface, three reasonably distinct regions of flow could be recognized [13].

p_{amb}, and $U_1 = 0$. Furthermore, for an ideal nozzle $W, Q, W_v = 0$. So computing the nozzle exit velocity from eequation 3.1:

$$U_i = \sqrt{2(h_1 - h_2)} \tag{3.2}$$

Now it is necessary to evaluate the enthalpy. It can be written as $h = c_p T$ where c_p is the specific heat at pressure constant and for air, with the hypothesis of ideal flow is about 1005 J/Kg K. From this, it is possible to evaluate the temperature as

$$\frac{T_1}{T_2} = (1 + \frac{\gamma - 1}{2} M_2^2) \tag{3.3}$$

the maximum velocity is calculated in the condition of choking flow, that since it is the condition of $M = 1$. So from this, it is easy to evaluate the value of T_1 and then U_i.

But this is not the real exit flow velocity, because for this the nozzle's losses are not negligible. So the most common way to compute the nozzle efficiency is to compute the so-called discharge coefficient, which considers also the compressibility of the flow and other losses such as skin friction in the nozzle:

$$C_D = \frac{1}{2} \frac{\rho U_j^2}{\Delta P_N} = \frac{U_j^2}{U_i^2} \tag{3.4}$$

where ΔP_N is the gauge pressure inside the nozzle and $C_D = 0.8$ Finally, it is possible to compute the value of the flow exit velocity, which will be an input parameter of the BLEW code.

But for this category of jet wiping, the value of the pressure gauge is about $\Delta P = 2 - 30 KPa$, as reported in the literature, so, in this case, the exit flow velocity will be strictly connected to it. Then, it is possible to compute the exit flow velocity using 3.4 as $U_j = 190 m/s$ and this is less than the max velocity to avoid the choking condition.

Moreover, it is possible to evaluate the wiping number, to have an idea of the wiping capabilities of the layout of the jet

$$\Pi_g = C_D \frac{\Delta P_n d}{\rho_l g Z^2} \tag{3.5}$$

This dimensionless parameter weights the pressure gradient generated by the gas with the hydrostatic pressure gradient of the liquid film. In this case, it leads to $\Pi_g = 3.3$, theoretically guaranteeing a discrete actuation capability over the liquid film.

3.3 Pressure and shear stress distribution

To model the height of the liquid film, firstly one must evaluate how the jet wiping interact with the liquid coat. To do this, an interesting approach is proposed by Beltaos [13], with an experimental facility to compute the relationship of shear stress and pressure distribution for circular jets impinging in a strip. They both depend on the value of stand-off distance H and the diameter of the nozzle d and for this kind of application it is $H/d \geq 6$, which is a compromise between wiping performance and the nozzle gas flow rate. Another crucial parameter that must be set up is the gauge of pressure in the nozzle, so the project variables are ΔP and H/D. A model of jet wiping is represented in Figure 3.3,

Firstly, assume the liquid film thickness h is much smaller than the stand-off distance, the pressure at the impinging point, or stagnation pressure is calculated, where the experimental relationship is

$$p_s = 50 \frac{\rho_g U_j^2}{2} \left(\frac{H}{d} \right)^{-2} \tag{3.6}$$

and the value of maximum shear stress

$$\tau_{0m} = 0.16 \rho_g U_j^2 \left(\frac{H}{d} \right)^{-2} \tag{3.7}$$

From this and the motion equation in the axial direction, it's possible to compute the expression of pressure distribution and shear stress distribution so that in the impingement region they fit closely to the curve given by

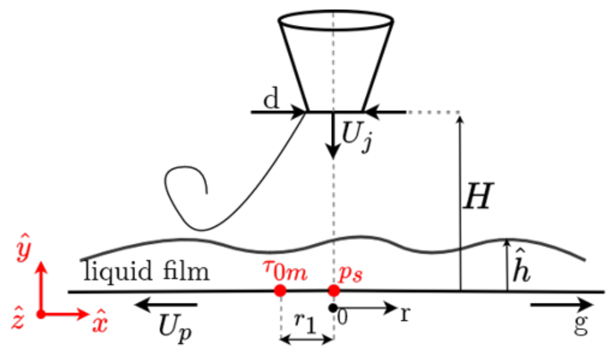


Figure 3.3: *Actuator model using in BLEW*
Schematic model of jet wiping, with the max pressure at impinging points and max shear stress in red

$$p_g = p_s e^{(-114(r/H)^2)} \tag{3.8}$$

and

$$\tau_g = 0.18\tau_{0m}\left(1 - e^{114(r/H)^2}\right)\frac{H}{r} - 9.43\frac{r}{H}e^{(r/H)^2} \tag{3.9}$$

From this relationship, the shear stress distribution and the pressure distribution, which represents the effective jet effect on the liquid film, can be derived. While the pressure distribution has its maximum at the point of impact, which is perpendicular to the nozzle, the shear stress is maximized at a certain point, namely at $r/h = 0.14$, as can be seen in the plot in Figure 3.4.

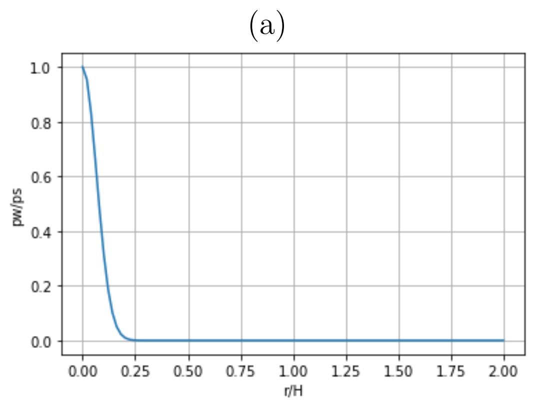
Therefore, it can be observed how the shear stresses have a great influence also outside of the impinging region, dictated by $r = 0.14H$. The suggested value for the dimensionless distance then depends on how strict the limit on the shear stress is.

At this point, it is possible to define the best option for the spacing and the nozzle diameter to obtain a correct pressure and shear stress distribution. In this case

$$H = 0.015m$$

$$d = 0.0015m$$

Once the relationship between shear stress and pressure distribution and the nozzle parameters are defined, it is possible to define the effect of the wiping jet for BLEW environment. Instead, the output of the ML algorithm is a value between 0 and 1 and therefore must be converted to a physical value. This is possible



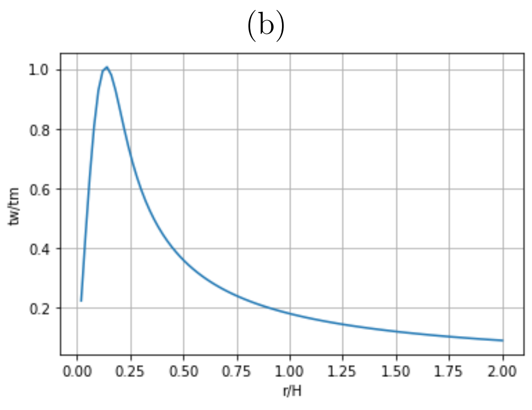


Figure 3.4: *Pressure and shear stress distribution*
Graphs illustrating the variation of dimensionless pressure (a) and shear stress (b) along the radial dimensionless distance are presented for BLEW. The highest magnitude occurs at the point of impingement in the pressure distribution, while the maximum value for the shear stress distribution is observed at a radial distance of $r = 0.14H$.

by normalizing the value for the maximum exit velocity and in this way simply evaluating the shear stress and pressure distribution to model the thickness of the fluid film as it is shown in Figure 3.5.

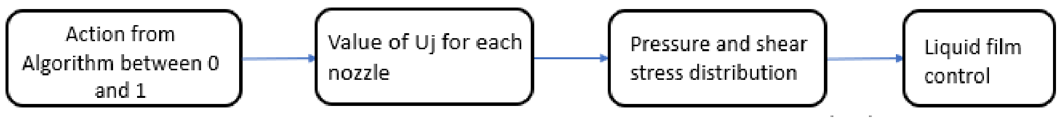

Action from Algorithm between 0 and 1
Value of Uj for each nozzle
Pressure and shear stress distribution
Liquid film control

Figure 3.5: *ML actions scheme*
How ML algorithm works to give actions to model the thickness of the liquid film, from its generated value between 0 and 1 to the interaction with the fluid.

Chapter 4

Optimization

In this section the goal is to analyze and optimize all the main features of BLEW, to obtain the best setup and then use the ML algorithm to learn the agent.

Let's start with a map that shows how BLEW code is structured as shown in Figure 4.1. In this case, the main is the environment, so it is composed of all the features of an ML env (step, render, action, cost function...)

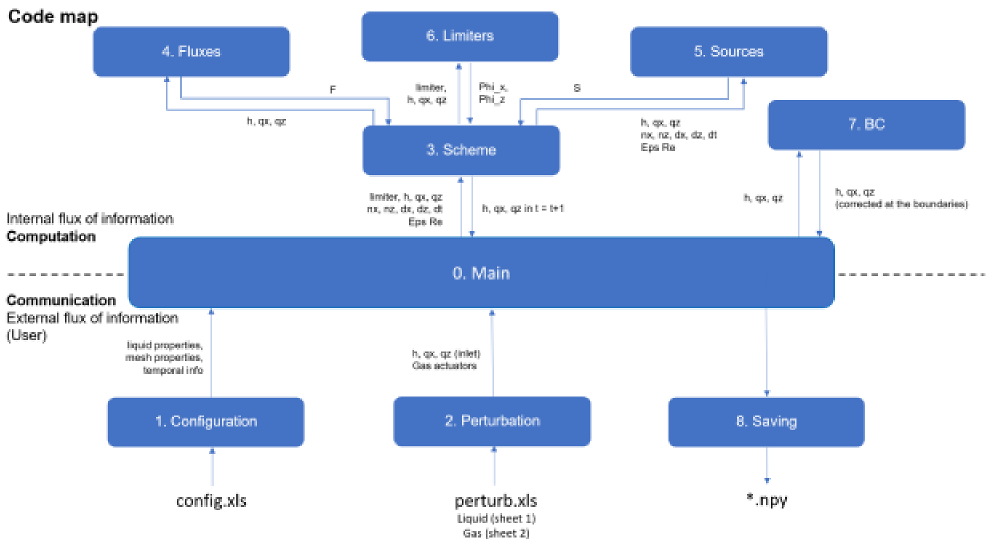


Figure 4.1: *Code Map*
How the code is composed, with the main code, the solver implemented with FV, the gas perturbation to compute the jets' actions, and other tools for writing and saving

Based on the wavelength of the perturbation, the domain size is defined along x and z, while the time step is based on the CFL value provided by the user and the value of the spatial discretization step and the velocity of the strip:

$$dt = \frac{CFL * dx}{U_p}$$

where $CFL = 0.5$

The primary aim of BLEW is to replicate these fluctuations in a finite domain, gauge the efficacy of machine learning algorithms in such an environment, and subsequently implement them in practical settings.

4.1 Observation points

The first step in optimization is to set the position of observation points. In BLEW they are represented by red points, but in the facility, they are physic sensors. They can be expensive and also they have a volume, so finding the best position can be useful in optimal control. Optimize jet position means that the goal is to give the controller much information about the state as possible, so they must give the controller the liquid film height as an input. It is a crucial part because from this the controller decides the action to give to the nozzle to reduce the instabilities that may appear. The number of sensors must be a compromise between the area covered and their volume and cost, so the number that can be chosen is 8 sensors. In the first setup, the 4 jets are positioned aligned with the jets as shown in Figure 4.2. It may be a good solution, but we have no confirmation that is the best. In order to enhance the efficacy of conducting a subsequent analysis, it is valuable to minimize the possibility of jets providing identical data to the controller. This scenario poses a challenge for the machine learning algorithm to effectively ascertain and determine the optimal course of action for all the jets involved. For this reason, a solution can be to analyze the correlation coefficient C_{ij}, which represents an index of how much these signals are related to each other.

$$C_{ij} = \frac{O_i' O_j}{\|O_i'\| \|O_j\|} \tag{4.1}$$

Where O is the vector of the observation for the episode length. If the correlation coefficient is 0, it means that signals are not correlated, otherwise if it is 1 they are fully correlated. Let's start with 1 line of 4 observation points as shown in Figure 4.2 and then analyze the signal through the correlation matrix that is shown in Figure 4.3.

Now is interesting to try another configuration. One solution may be to use a Latin hypercube distribution, a particular distribution implemented in PyDOE [14] a Python library. Latin Hypercube Sampling (LHS) is a way of generating random samples of parameter values. It is widely used in Monte Carlo simulation because

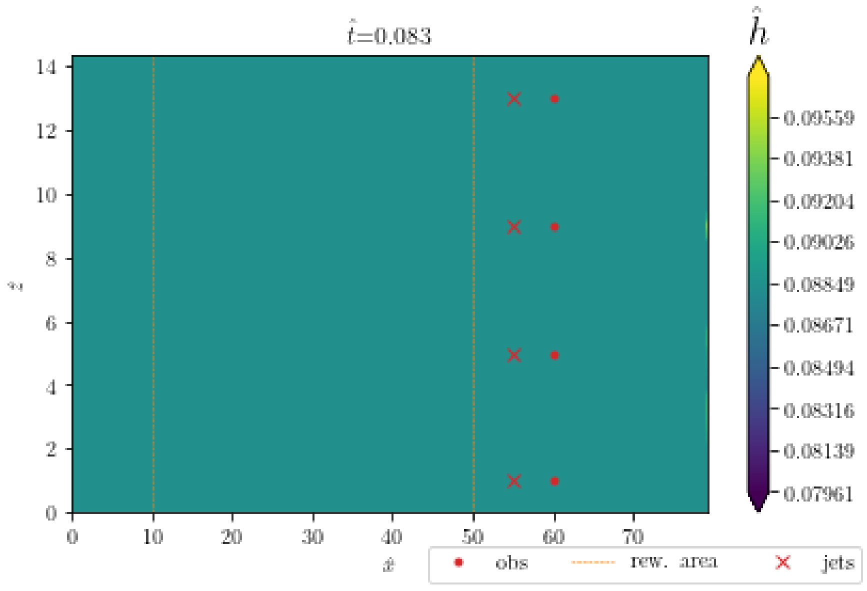
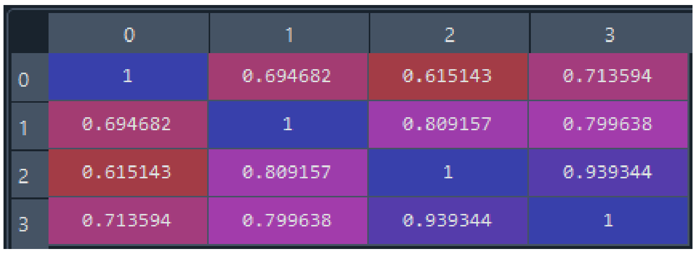


Figure 4.2: *Default set-up*
Initial set-up with 4 obs points aligned and 4 jets aligned



	0	1	2	3
0	1	0.694682	0.615143	0.713594
1	0.694682	1	0.809157	0.799638
2	0.615143	0.809157	1	0.939344
3	0.713594	0.799638	0.939344	1

Figure 4.3: *Correlation Matrix of 4 obs points aligned*
Correlation numbers are so high, so signal are correlated with each other

it can drastically reduce the number of runs necessary to achieve a reasonably accurate result. LHS is based on the Latin square design, which has a single sample in each row and column. A “hypercube” is a cube with more than three dimensions; the Latin square is extended to sample from multiple dimensions and multiple hyperplanes. Such configuration is similar to having N rooks on a chess board without threatening each other. Let’s try with an LHS such as Figure 4.4

In this case with 8 obs points, the correlation matrix will be an 8x8 as in Figure 4.5

It can be easy to see that the correlation coefficient is so close to 0, and they are more minor than the matrix with 4 points. This implies that in this scenario, it becomes feasible to provide the controller with additional data regarding the

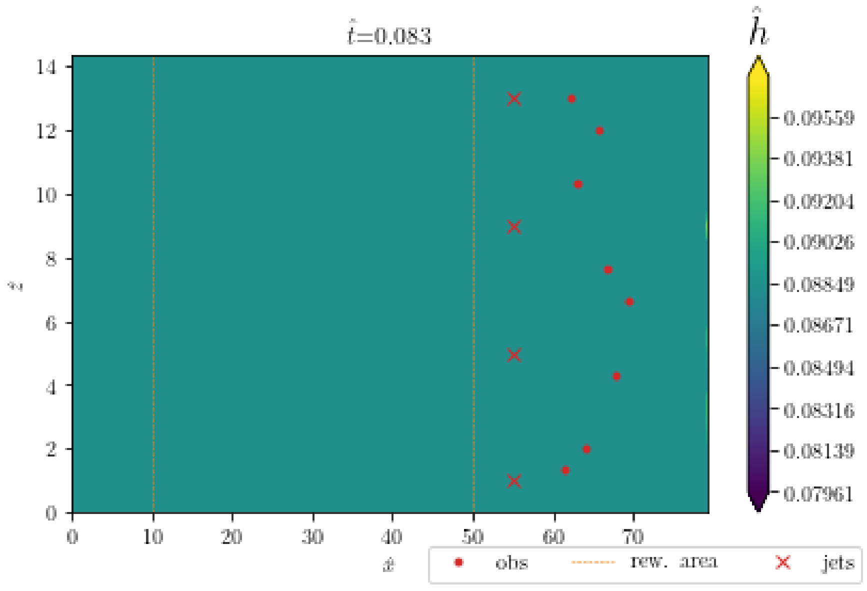
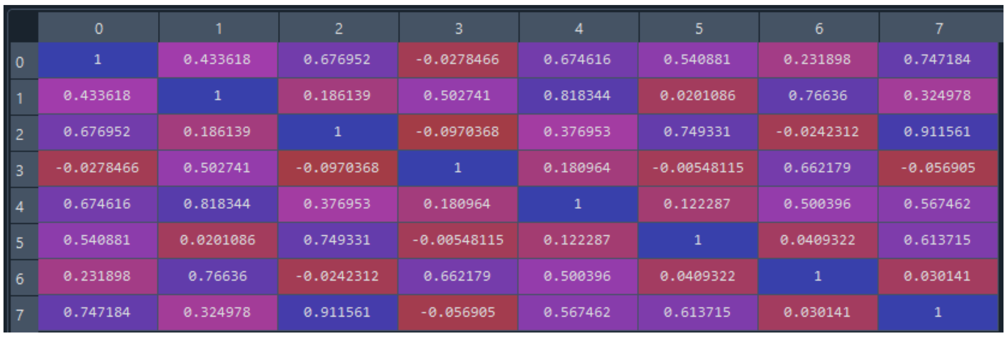


Figure 4.4: LHS distribution
New set-up with LHS distribution of obs points to optimize the correlation coefficient



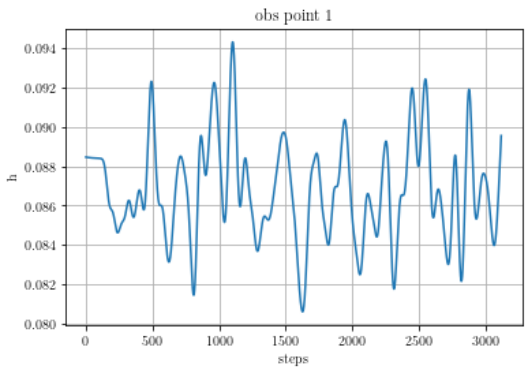
	0	1	2	3	4	5	6	7
0	1	0.433618	0.676952	-0.0278466	0.674616	0.540881	0.231898	0.747184
1	0.433618	1	0.186139	0.502741	0.818344	0.0201086	0.76636	0.324978
2	0.676952	0.186139	1	-0.0970368	0.376953	0.749331	-0.0242312	0.911561
3	-0.0278466	0.502741	-0.0970368	1	0.180964	-0.00548115	0.662179	-0.056905
4	0.674616	0.818344	0.376953	0.180964	1	0.122287	0.500396	0.567462
5	0.540881	0.0201086	0.749331	-0.00548115	0.122287	1	0.0409322	0.613715
6	0.231898	0.76636	-0.0242312	0.662179	0.500396	0.0409322	1	0.030141
7	0.747184	0.324978	0.911561	-0.056905	0.567462	0.613715	0.030141	1

Figure 4.5: Correlation Matrix LHS
In this case, opposite figure 4.3, the correlation coefficient are small, and most of them are close to zero

condition, as evident from the graphical representation of the signal of h at two specific sampling instances, as depicted in Figure 4.6.

But before giving this information about the state to the ML algorithm, is important to normalize these values. To learn well, the policy needs observation as a value between 0 and 1, so, in this case, is important to define an observation space functional for the ML policy. An effective approach to standardize the observations is by utilizing the highest and lowest values derived from perturbation inlet realistic.

(a)



(b)

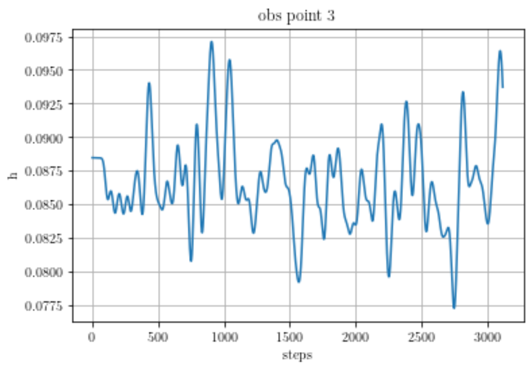


Figure 4.6: *Comparison between the signal in 2 obs points*
Plot of the signal of liquid film thickness in 2 of the new obs points.

After acquiring these values, it is possible to normalize them as

$$obs[i] = \frac{obs[i] - min(obs)}{max(obs) - min(obs)} \tag{4.2}$$

Now all the ingredients for the obs point of the environment are defined and they are ready to pass to the controller.

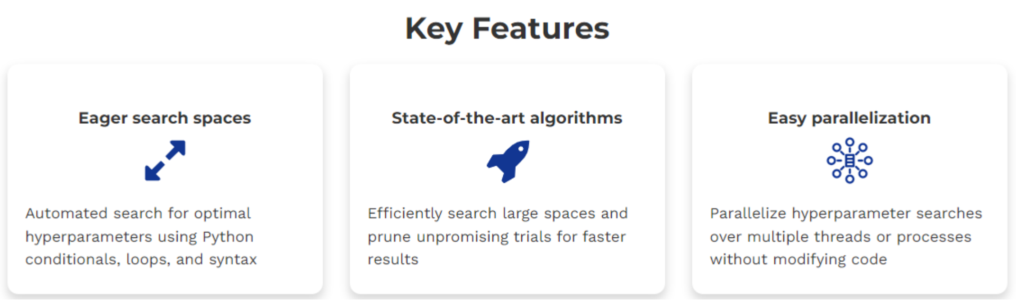
4.2 Hyperparameters optimization (HPO)

Hyperparameters optimization (HPO) is the process of selecting the optimal values for the hyperparameters of a machine learning model to achieve better performance. In the context of reinforcement learning (RL) algorithms, HPO refers to the process of selecting the optimal hyperparameters for the RL algorithm to improve its learning and decision-making capabilities.

RL algorithms have several hyperparameters that can significantly affect their performance, such as the learning rate, discount factor, exploration rate, batch size, and neural network architecture. HPO methods aim to find the best combination of hyperparameters that optimize the performance of the RL algorithm on a specific task.

There are various HPO techniques available for RL algorithms, such as grid search, random search, Bayesian optimization, and evolutionary algorithms. These techniques search through the hyperparameter space and try different combinations of hyperparameters to identify the best set of hyperparameters that optimize the RL algorithm's performance on a given task.

For the HPO it can be used a library called Optuna [15], which is an optimizer based on Bayesian Optimization to optimize hyperparameters of the reinforcement learning algorithm. It has some features that make it more efficient applied to BLEW such as easy parallelization and efficient and fast trial search as shown in Figure 4.7.



Key Features

Eager search spaces

Automated search for optimal hyperparameters using Python conditionals, loops, and syntax

State-of-the-art algorithms

Efficiently search large spaces and prune unpromising trials for faster results

Easy parallelization

Parallelize hyperparameter searches over multiple threads or processes without modifying code

Figure 4.7: *Optuna features*
By leveraging the features offered by Optuna, we can efficiently tune the hyperparameters of the reinforcement learning algorithm for flow control, ultimately leading to improved performance and better convergence. [15].

HPO is crucial in RL because the optimal set of hyperparameters varies depending on the task and environment, and a suboptimal set of hyperparameters can lead to poor performance or slow learning. Therefore, optimizing hyperparameters is essential to achieve the best performance of RL algorithms in various applications.

In this case, finding the best trade-off of hyperparameters for RL algorithms such as PPO2 and DDPG applied on BLEW can influence the capabilities to learn of the agent because they are set as input of the model. So the goal is tuning Hyperparameters to find the best setup to optimize the learning process in terms of time and performance

In PPO2 the main hyperparameters that need to be changed from the default, are

- discount factor γ

  It usually is a value between 0.9 and 0.99 and it is an index of how much the new action in the present is influenced by the past.

- n steps

  It is a parameter that denotes how many steps the policy is updated changing the weights of the neural network. If it is too small, the policy may have not enough information to update well, on the other hand, if it is too high the computational cost increases, and also the training can become unstable without converging. It is usually about the same length as one episode.

- learning rate

  It decides how much to update the weights of the neural network. It can be between 0 and 1: if it is 1 the algorithm can learn quickly, but it is more unstable with a lot of entropy loss. If it is too slow, the algorithm may take too much time to learn.

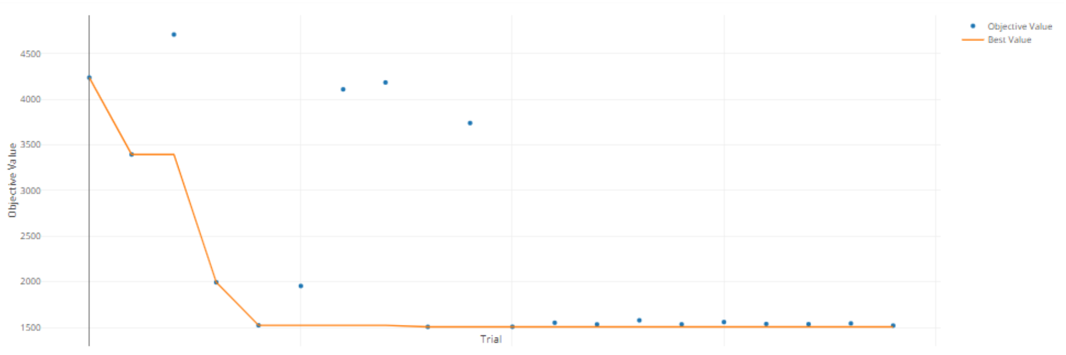
- entropy coefficient

  in RL, entropy refers to the predictability of the actions of an agent. This is closely related to the certainty of its policy about what action will yield the highest cumulative reward in the long run: if certainty is high, entropy is low, and vice versa.

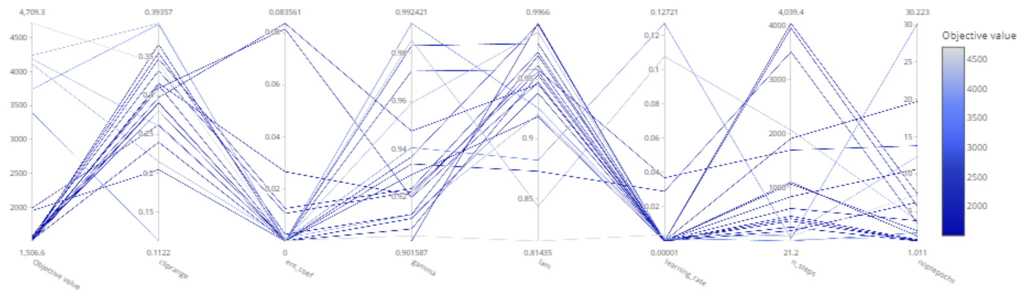
In this case it is possible to set the number of trials and the number of training for each agent trial. For this kind of RL algorithm, the objective value is the reward from the cost function, so BO tries to set the hyperparameters to minimize it. In the end, is it possible to show the results through the Optuna dashboard to see all the parameters and how they change during this training as in Figure 4.8.

Let s compare now the learning curves of a selected setup using several hyperparameters as shown in Figure 4.9. The first one is done using default hyperparameters from the library stable baselines while the second is done after HPO. The notable enhancement in the reward is associated with modifying the hyperparameter *nsteps*, which could potentially be of a similar value as the duration of the episode.

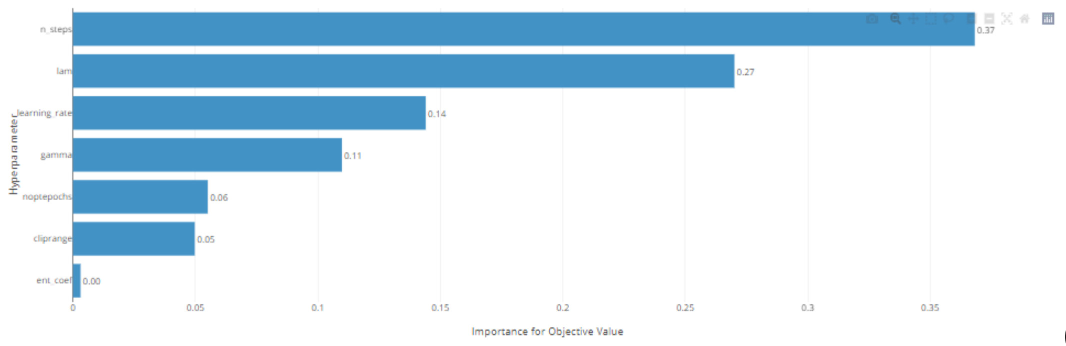
From this learning curve is clear that the ability to learn of the policy increase changing hyperparameters, as we expected, and for example, in this case, the performance of PPO2 is improved by about 5% evaluated on 1000 episodes.



(a)



(b)



(c)

Figure 4.8: *Optuna HPO*
In figure (a) how Optuna has worked to optimize reward changing hyperparameters, in (b) how hyperparameters change in each trial, in (c) how each hyperparameter influences the reward

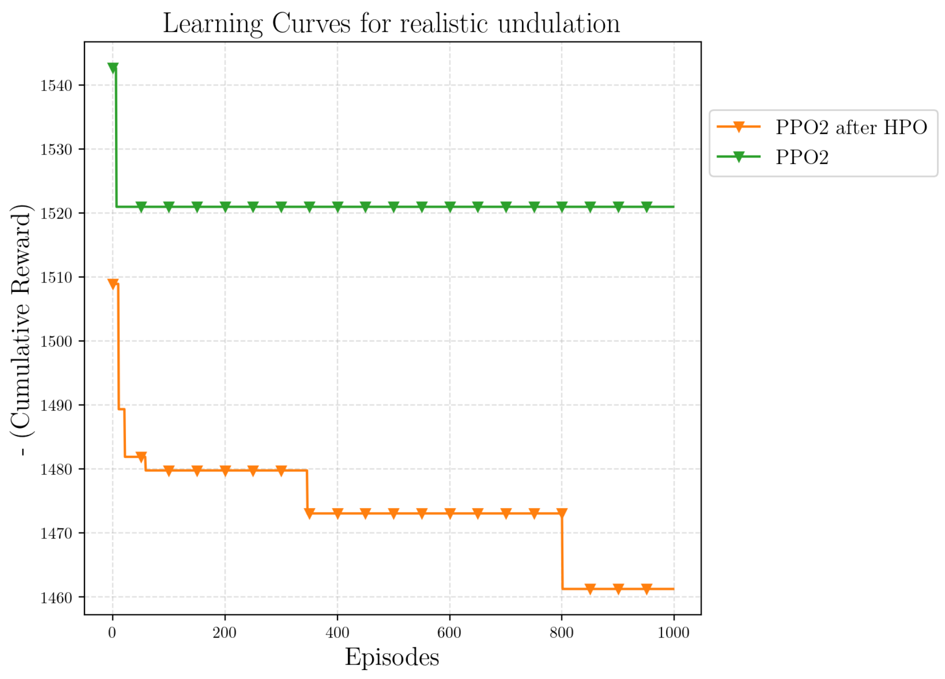


Figure 4.9: *PPO2 HPO conf*
Comparison in terms of learning curves for a PPO2 before and after the HPO. The reward after HPO is much better.

4.3 Artificial Neural Network (ANN) size

This type of optimal control works with a neural network. To explain how it works, you can imagine that they are similar to the neurons that make up the human body. It is inspired by the way the human brain processes information: it's made up of layers and neurons, and they decide on the information for each learning step, with some weights updated at each step. The network takes an input signal and applies a series of mathematical operations to it, passing it through the layers of neurons until it produces a predictable signal output. An example of a neural network with 3 hidden layers and 9 neurons each one is shown in Figure 4.10.

In this case, it is important to find a good compromise between the number of layers and the number of neurons: If the number of neurons is too high, the ML algorithm can learn well, but the computational cost increases dramatically. Therefore, a good compromise is to use a neural network with 4 layers, with 64, 128, 128, and 64 neurons in each layer.

It means that we have 4 layers for the actors and the oracle with the same number of neurons, which take as input the obs, and then they interface respectively with the action and the value.

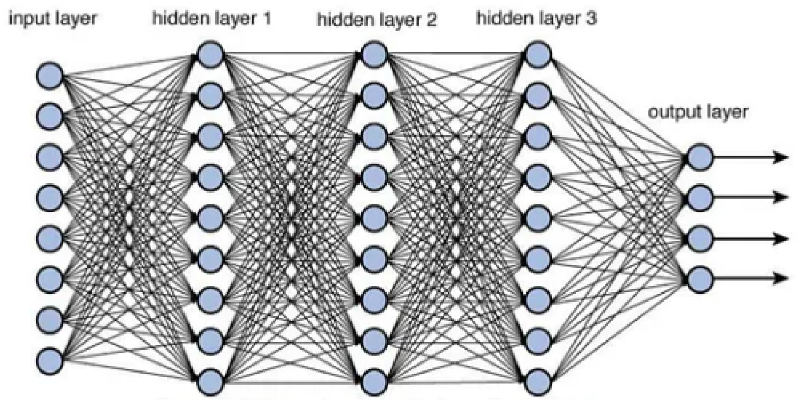


Figure 4.10: *Neural Network*
From the input layer, the information passes through 3 hidden layers until arriving at the output layer. Each layer is composed by 9 neurons [16]

Another point to improve policy performance is to use regularization for the neural network. In this case it may be useful to use the L2 regularization, also known as weight decay, which is a common technique used in machine learning, to prevent overfitting and improve generalization. It works by adding a regularization term to the loss function, which encourages the model to have smaller weights. This can be applied to the neural network parameters to control the complexity of the model and reduce overfitting. The regularization term is added to the original objective function being optimized during the training process.

4.4 Jet position

Now that the main features of the ML algorithm such as hyperparameters and neural network size have been determined, it is time to define the position of the nozzles. Using the Beltaos distribution for impinging circular jets as in Chapter 3, it is feasible to calculate the shear stress and pressure distribution, and subsequently observe how the outcomes vary in relation to the positioning of the jets in terms of learning performance. For this reason, various experimental arrangements can be tried as shown in Figure 4.11, and the interference of each ray with the others must also be considered. One possibility is to work with alternating jets in two lines to solve the problem of interference with an opposite control, but when using, for example, 7 jets, there is still the problem of high computational costs.

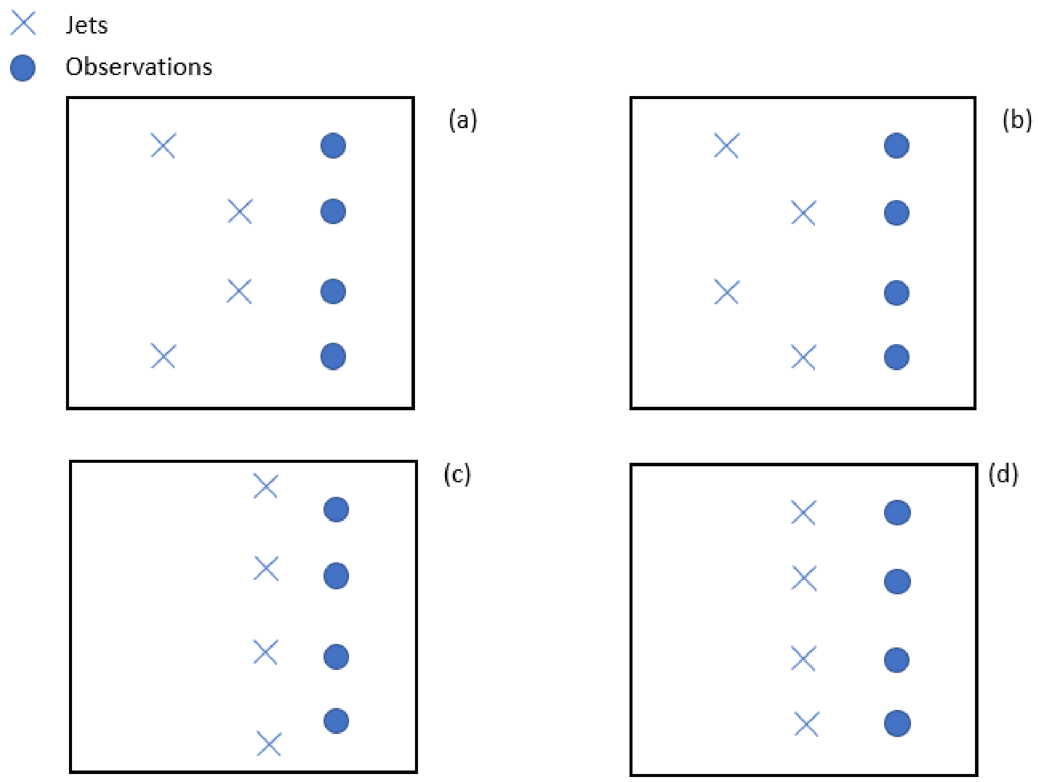


Figure 4.11: *Jets' position*
Multiple nozzle configurations can be examined to assess various arrangements aiming to enhance the overall performance outcome.

The efficiency of that configuration is evaluated in terms of learning curves, that represent the ability of the policy to learn and then to give the best reward. In this case, is possible to compare the learning curves for each jet setup, and then evaluate the final reward. After some episode of training is clear that the best choice is to use jets closer than the original configuration. The distance between each jet should be at least the distance that permits the pressure and shear stress distribution to not interfere with the others. For this reason, the distance is estimated to have $p_w = p_{amb}$ and $\tau = 0.2\tau_{max}$. Known distribution, it is not difficult to evaluate the point that satisfies this condition, and after that to convert the nominal value in dimensionless coordinates for BLEW code.

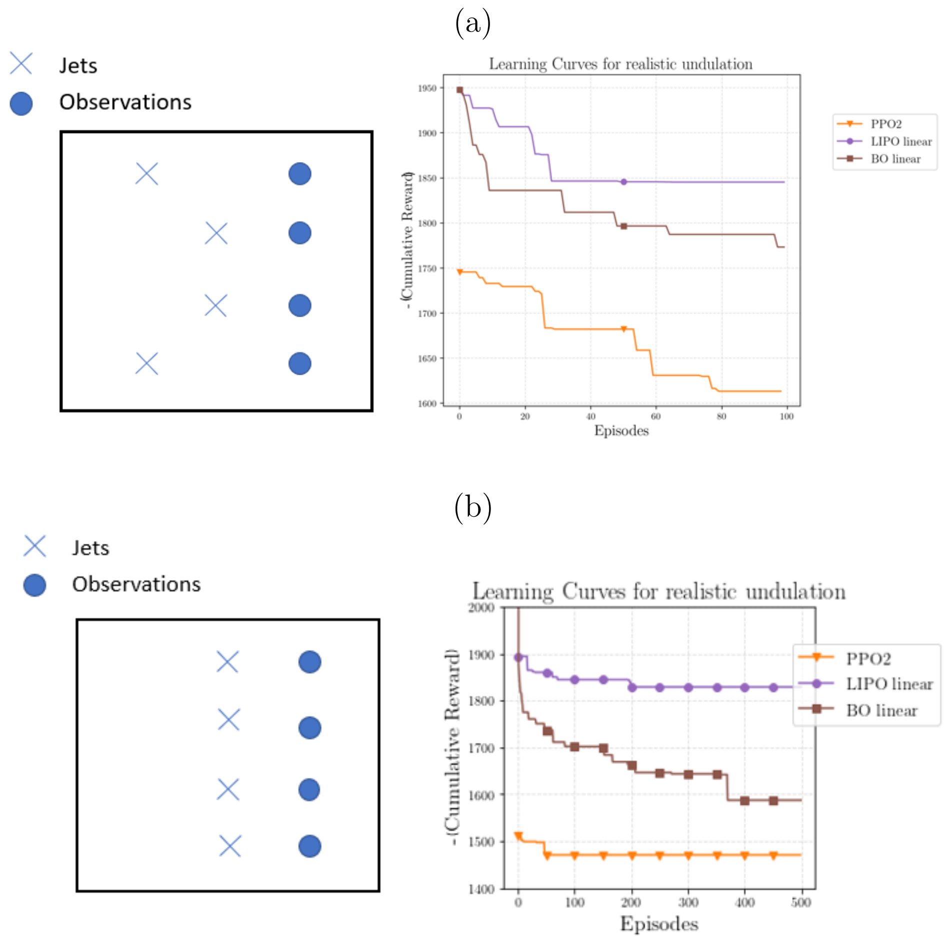


Figure 4.12: *Comparison of learning outcomes between two configurations In the first setup (a), the level of reward is minimal, whereas in the optimal configuration (b), we experience a remarkable improvement after just 100 episodes of learning.*

We compare different configurations based on training performances over 100 episodes. It is clear to see that with the last configuration, the gain is higher than the other, with a performance improvement of more than 10% using the BO and about 8% using the PPO2 as shown in Figure 4.12.

These sims are made with a very low number of episodes because they are used just to choose the best setup for made longer simulations using all the improved performances.

4.5 Reward area

Another crucial part of the machine learning algorithm that needs to be improved is the reward area, which is a part of the domain where the reward is evaluated. So in this case it represents the area where the sensor takes the info about the thickness of the liquid film after the control of jets. The goal is to maximize the reward of the cost function in this area. One of the main problems in this numerical simulation is the numerical dispersion, so the wave may be auto-dumped by itself. So, if the reward area is too far from the jets, it is clear that it is not possible to evaluate the performance of ML because in this case, we are evaluating not the effect of the jet wiping but the effect of the numerical dispersion. For this reason, considering the domain's size, it is important to consider the reward area from about the position of jets, where the perturbation starts, to a point before the end of the domain. For this reason, the reward area is chosen in a strategic position that is $10 < \hat{x} < 50$.

4.6 Multiprocess environment

These fluid dynamics simulations are very expensive for computational costs. For this reason, one solution to reduce the time is to use a multiprocess custom environment: in this way is possible to run in parallel more episodes, and, depending on the number of CPUs that are used, reducing dramatically the time of sims. The scheme of how the ML algorithm with a multiprocess environment works is shown in Figure 4.13.

But using a vectorized env present some advantage and some disadvantages

Advantages:

- Increased efficiency

  With multiple processes running in parallel, the RL agent can interact with the environment and collect experience data more quickly. This can significantly speed up the learning process compared to a single-process environment.

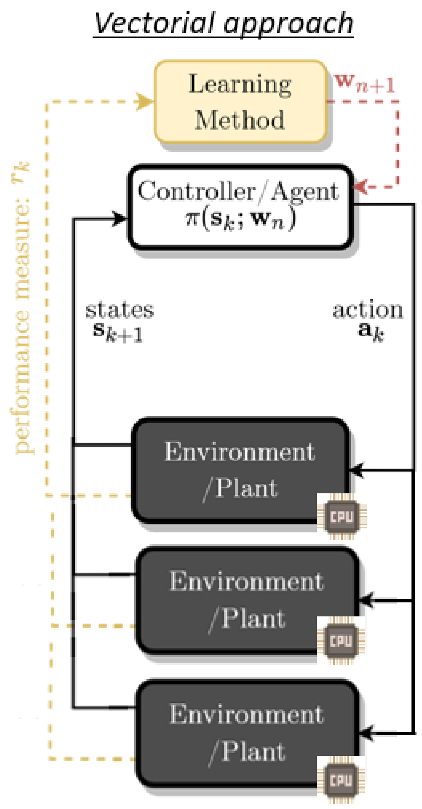


Figure 4.13: *Multiprocess scheme*
How algorithms operate within the vectorized environment: Multiple environments are executed simultaneously on each CPU, and the action vector a_k dictates their respective actions.

- Enhanced exploration

  In RL, exploration is crucial for discovering optimal policies. Multi-process environments can enable different instances of the RL agent to explore different parts of the environment simultaneously, leading to more diverse experiences and potentially better policy exploration.

- Utilization of multi-core systems

  Modern computers often have multiple CPU cores, and multi-process environments can utilize these cores effectively. By distributing computation across cores, the RL algorithm can take advantage of parallelism and achieve faster training.

Disadvantages:

- Increased complexity

  Managing multiple processes introduces additional complexity to the RL algorithm. Coordinating data sharing, synchronization, and communication between processes requires careful implementation and can be error-prone. Debugging and maintaining a multi-process environment can be more challenging.

- Communication overhead

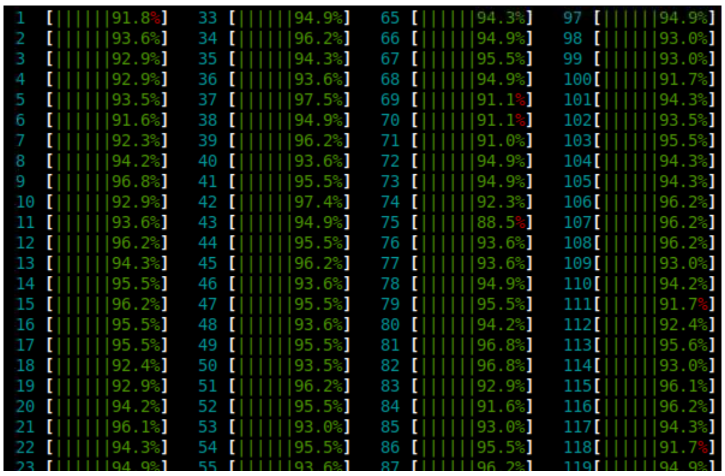
  Inter-process communication overhead can be a bottleneck in multi-process environments. Sharing information between processes may incur delays, reducing the overall efficiency gained from parallelization.

- Synchronization issues

  In RL, it is often necessary to synchronize different processes to update the agent's policy or value functions. Ensuring proper synchronization and avoiding race conditions can be non-trivial, especially when dealing with complex RL algorithms or distributed systems.

- Resource utilization

  Multi-process environments consume more system resources, including CPU, and memory. If the RL algorithm and the environment are not efficiently designed or if the system lacks sufficient resources, the overall performance may be compromised.



```
1   [||||||91.8%]   33 [||||||94.9%]   65 [||||||94.3%]   97 [||||||94.9%]
2   [||||||93.6%]   34 [||||||96.2%]   66 [||||||94.9%]   98 [||||||93.0%]
3   [||||||92.9%]   35 [||||||94.3%]   67 [||||||95.5%]   99 [||||||93.0%]
4   [||||||92.9%]   36 [||||||93.6%]   68 [||||||94.9%]   100[||||||91.7%]
5   [||||||93.5%]   37 [||||||97.5%]   69 [||||||91.1%]   101[||||||94.3%]
6   [||||||91.6%]   38 [||||||94.9%]   70 [||||||91.1%]   102[||||||93.5%]
7   [||||||92.3%]   39 [||||||96.2%]   71 [||||||91.0%]   103[||||||95.5%]
8   [||||||94.2%]   40 [||||||93.6%]   72 [||||||94.9%]   104[||||||94.3%]
9   [||||||96.8%]   41 [||||||95.5%]   73 [||||||94.9%]   105[||||||94.3%]
10  [||||||92.9%]   42 [||||||97.4%]   74 [||||||92.3%]   106[||||||96.2%]
11  [||||||93.6%]   43 [||||||94.9%]   75 [||||||88.5%]   107[||||||96.2%]
12  [||||||96.2%]   44 [||||||95.5%]   76 [||||||93.6%]   108[||||||96.2%]
13  [||||||94.3%]   45 [||||||96.2%]   77 [||||||93.6%]   109[||||||93.0%]
14  [||||||95.5%]   46 [||||||93.6%]   78 [||||||94.9%]   110[||||||94.2%]
15  [||||||96.2%]   47 [||||||95.5%]   79 [||||||95.5%]   111[||||||91.7%]
16  [||||||95.5%]   48 [||||||93.6%]   80 [||||||94.2%]   112[||||||92.4%]
17  [||||||95.5%]   49 [||||||95.5%]   81 [||||||96.8%]   113[||||||95.6%]
18  [||||||92.4%]   50 [||||||93.5%]   82 [||||||96.8%]   114[||||||93.0%]
19  [||||||92.9%]   51 [||||||96.2%]   83 [||||||92.9%]   115[||||||96.1%]
20  [||||||94.2%]   52 [||||||95.5%]   84 [||||||91.6%]   116[||||||96.2%]
21  [||||||96.1%]   53 [||||||93.0%]   85 [||||||93.0%]   117[||||||94.3%]
22  [||||||94.3%]   54 [||||||95.5%]   86 [||||||95.5%]   118[||||||91.7%]
23  [||||||94.9%]   55 [||||||93.6%]   87 [||||||96.2%]   119[||||||94.9%]
```

Figure 4.14: *Multiprocess Env running in 100 cpu*
All cores of the machine runs for the vectorized env

It is also important to find a good compromise between the time of simulations and machine occupation because if the computer is saturated, the processor starts to run in parallel, and the runtime increase. Figure 4.14 is shown the processors of the computer which are running to solve a multiprocess environment.

For this kind of environment, a good strategy of optimization can be to use a random environment, in which each episode starts from a random point of the simulation; in this way, it is possible to maximize the efficiency of the multiprocess because each core can have its env different from the others and the policy takes information from all of them.

Chapter 5

Cost function and actions

For an ML policy is fundamental to define as well as possible the cost function. It gives the reward to the algorithm, so if it is not definite correctly the algorithm can't learn about the state of the system or it can't understand how to work to optimize it. In the same way, a crucial part is the choice of the actions to give to jets to regulate the shear stress and pressure distribution.

5.1 Actions

The actions represent the choices that an RL agent can take in a given state of the environment and through it interacts and influences the environment. First of all, is crucial to define a correct action space. The action space refers to the set of all possible actions that the agent can choose from. It can be discrete, where there is a finite number of actions, or continuous, where actions lie within a continuous range. In BLEW, the action space is continuous. Actions from ML are between 0 and 1, but choosing directly these values as boundary limits for the action space can cause some instability problems, mainly for the PPO2, which is parametrized by a Gaussian so can in theory output infinite values. In practice, the action is clipped to match the boundaries but because of that and the initialization, it is recommended to do the rescaling inside the environment and keep the bounds in [-1, 1].

After that, can be useful to use a filter for the action, to have more smoothed actions, and to delete some peaks. To do this, a good strategy is to use the Exponential Moving Average (EMA) that considers not only the action at time step t but also the previous actions. In other words, it is like giving more importance to the last experience or memories than to older ones, assuming those are represented by data points.

$$\hat{y}_{T+1|T} = \alpha y_T + \alpha(1-\alpha) y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + \cdots \ , \tag{5.1}$$

where y is the action and α is the smoothing parameter between 0 and 1 and is the only controlling parameter that controls the smoothness of the fitted curve so it can be used as a filter. In this case the smoothing parameter is $\alpha = 0.5$. Consequently, this filtering process enables the controller to operate unrestrictedly within the desired frequency range for addressing the control issue while safeguarding the stability of the numerical solver by averting sudden alterations.

5.2 Cost function

- **Variance**

  The first idea is to use the variance to evaluate how much in each episode the thickness of the liquid film, the h, is far from the desired value, which in this case is the h's value at the inlet. For this reason, the first approach may be to use the variance, which is defined as

  $$R = \frac{1}{N} \sum_{i=1}^{N} (h - h_0)^2 \tag{5.2}$$

  where h is the thickness of the liquid film in each point, h_0 is the thickness at the inlet and N is the number of points in the reward area. This value is summed for all the points in the reward area, and then multiplied for -10^3 to have more sensibility in the reward; RL algorithm has a maximize direction, so the reward is a negative number. The final goal is to have the greatest possible reward, and to do that the policy has to minimize the standard deviation in each episode, through jets; working on shear stress and pressure distribution policy can learn that to reduce the reward the best option is to have a more flat substrate of film, so it mince that the value of $h - h_0$ is more close to 0; in this way the policy can give to the jets the best action to do.

  Minimizing variance as a reward can simplify the control problem by focusing on a single metric. Instead of dealing with multiple conflicting objectives or constraints, the agent's objective becomes straightforward: to reduce variance, that mince reduces fluctuations and disturbances within the film. This simplification can make the learning process more tractable and improve the interpretability of the control policy. This is why in this case the variance can be the best choice.

- **Variance and mean**

  Using just the variance, the ML algorithm should not understand what it must do well. For this reason, a good strategy to define the cost function is to use both variance and mean, to give more sensibility to the system and try to help the policy to learn. In this case, the reward can be defined as

  $$R = \frac{\sigma^2}{\bar{h}} \tag{5.3}$$

  But if the sensibility of the reward increase, on the other hand, the algorithm has two directions of optimization, the variance and the mean. This may be a problem in such a simulation because it can optimize just the direction of the mean and not the variance, and in this case, the result is just to wipe more, and it is not the expected result. In this case, a sensibility analysis may occur, to be sure that both variance and mean have the same influence on the reward and that they are about of the same order. But in this case, it is very hard to learn for an ML algorithm and it may take more episodes than the variance.

Chapter 6

Results

6.1 Undulation control

In this section, we showcase the results of the diverse control algorithms concerning their learning progress graphs and control actions for the liquid film instabilities. Considering the heuristic nature of these control strategies, we conducted multiple training sessions for each approach, employing various seed values for the random number generator. The learning curve is defined as the maximum cumulative reward R achieved in each episode across the different training sessions.

As said before, this thesis aimed to control the instabilities called undulation, that in BLEW are represented by waves. So using empirical shear stress and pressure distribution induced by the nozzle, the goal is to smooth as much as possible this wave to have a more laminar substrate. To do this, a good strategy is to compare the reward after training the ML agent with the reward in the uncontrolled condition, which minces with action jets null. After defining the jets and obs points position, let's start to see the results, plotting the learning curves and the gif in 2D and 3D .

In this case, the cost function is defined as said in Chapter 5, so it is negative; it minces that if the ML algorithm works well, it should maximize the reward, starting from a random value and trying to arrive at to value close to 0.

The reward is given by the cost function as a sum of the variance in the reward area. In the case of the uncontrolled condition, the results are

$$R = -1906$$

So in the following graph, it is possible to see how the reward goes to reduce with a different kind of ML algorithm. Each episode is composed of 3120 timesteps and the environment can be a serial deterministic environment, which is used in serial simulation, or randomic environment, which is more useful in a multiprocess environment.

After a phase of learning each agent trained is tested again in an evaluation episode, which consists in load the model learned in a new environment. If the reward, in this case, is better than the uncontrolled condition, in mince that it has learned and the simulation has gone as expected. It can be possible because after a phase of learning all the features of the policy, such as neural network weights and control low, are saved in a file that is the output of all the ML algorithms.

6.1.1 Learning on 100 episodes

Following the learning phase, the rewards obtained in each episode are recorded in a monitoring system, allowing for the visualization of learning curves. These curves provide valuable insights into the performance of the learning process. By comparing different algorithms, one can discern which algorithm exhibits the highest efficiency in terms of learning and computational cost. Let's start to see the LC with 100 ep for all kinds of algorithms presented in Chapter 2 (Figure 6.1)

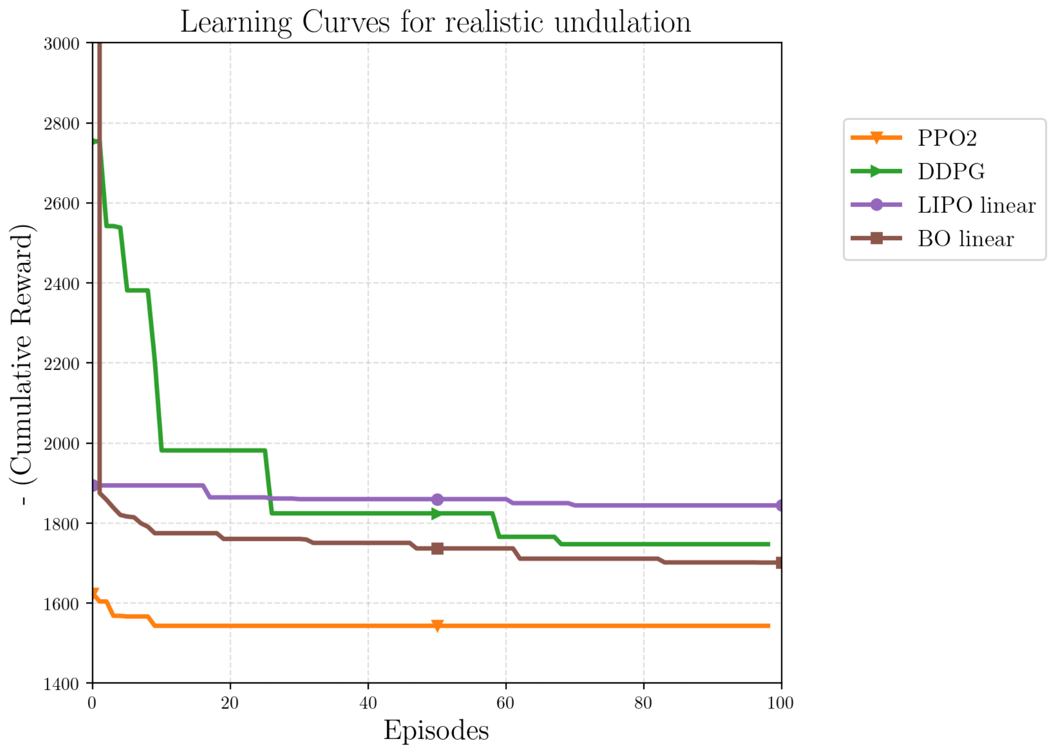


Figure 6.1: *LC 100 episodes*
Learning curves with upper bounds for LIPO, BO, DDPG, and PPO2 evaluated on 100 ep of training

This LC is plotted with the upper bound strategy, which minces that the lower bound of a set of numbers is the smallest number in the set represented. From this plot, the best result is the PPO2, and in this case, is

$$R = -1543$$

so it means that in this case, the gain compared with the uncontrolled condition is about 19% of the reduction of instabilities.

But before going ahead is important to do a premise. In the field of reinforcement learning, the primary objective of an agent is to acquire knowledge on how to make optimal decisions by gathering and analyzing data through interactions with its environment. To achieve a high-performing decision-making strategy, it is crucial to accumulate data that exhibits sufficiently favorable behavior, which includes sequences of states, actions, and corresponding rewards. One prevalent and straightforward approach to enhance the quality of collected data involves introducing randomness into the action selection process, called action noise. By injecting noise into the decision-making mechanism, the agent explores a broader spectrum of action sequences, ultimately leading to the discovery of valuable insights and knowledge. All these RL algorithms are implemented with the action noise so the results are also influenced by them and they may have a random part that is more significant when the number of episodes is not too high. In this case, the noise used is the time-correlated Uhlenbeck-Ornstein noise which is the most useful in this application.

Let's continue plotting the learning curves without upper bounds, to see the ability to learn the ML algorithm as in Figure 6.2. In this case, it's interesting to plot the DDPG, which is clear that starts from a very high and random value and that it decreases each timestep, to represent that the policy is effectively learning.

But to evaluate how the policy is learning, a good strategy can be to see the parameters through a tensorboard (Figure 6.3), where it is possible to visualize for the DDPG the actor loss and the critic loss. They are two indexes and the actor loss is typically defined as the negative expected value of the critic's Q-value for the chosen action while the critic loss is computed by comparing the predicted Q-value with the target Q-value, which is usually obtained using the Bellman equation. While the critic is going close to zero, the actor loss increase in each episode, and this is a clear signal that the learning is not completed.

To see how the waves each step can be reduced, it can be possible to make a gif for an episode(Figure 6.4). To evaluate the performance of the ML algorithm, we test the trained model on a single episode, called evaluation episodes, where it is possible to see the improvement.

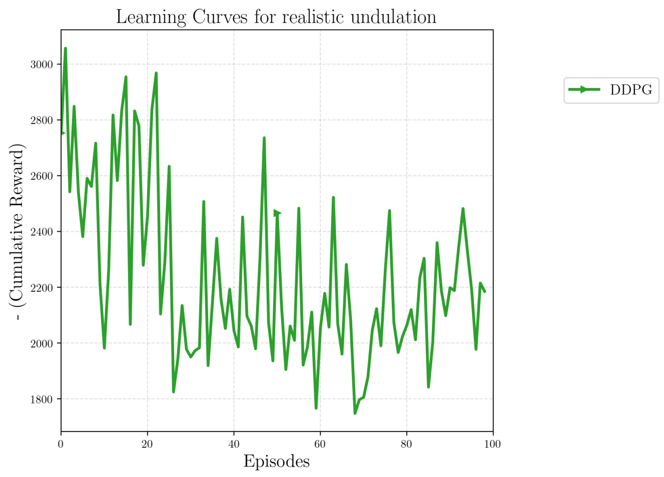


Figure 6.2: *LC DDPG 100 ep*
learning curves without upper bound for DDPG. It is evident the descendent trend because starting from a random point it is learning

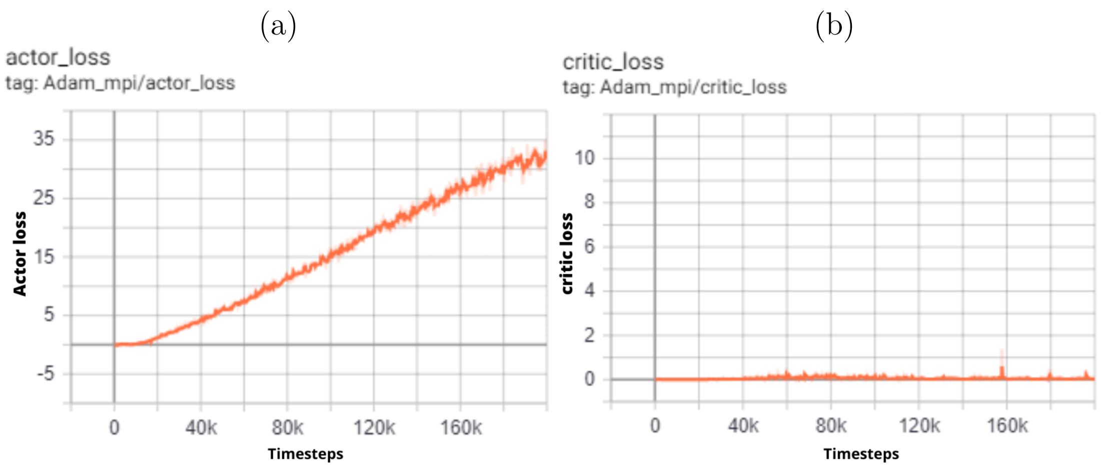
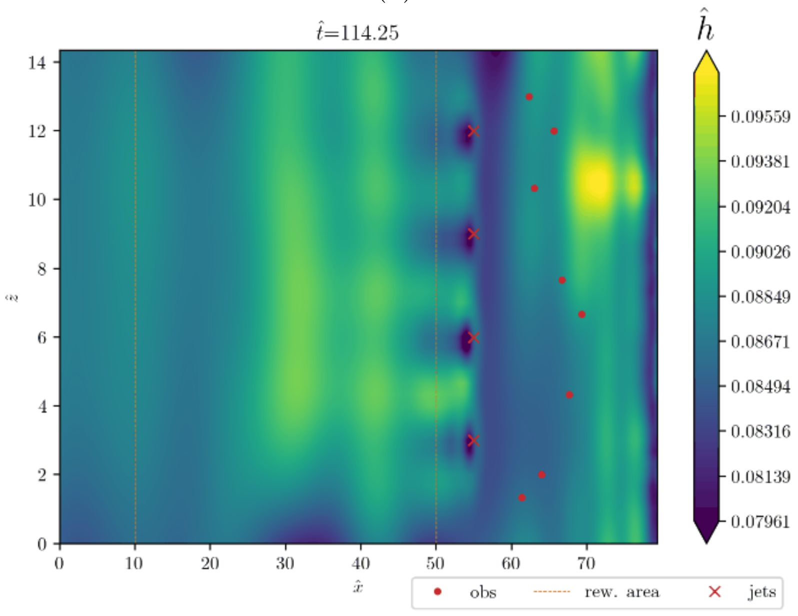


Figure 6.3: *Actor and critic loss for ddpg 100 episodes*
While the critic is performing well because its loss is close to zero, the actor loss increases to represent that the learning is not finished

And it can be interesting also to plot a 3D visualization as in Figure 6.5, where it is clear to see that the perturbation at the inlet is smoothed by the control jets, represented in the plan upstairs.

(a)



(b)

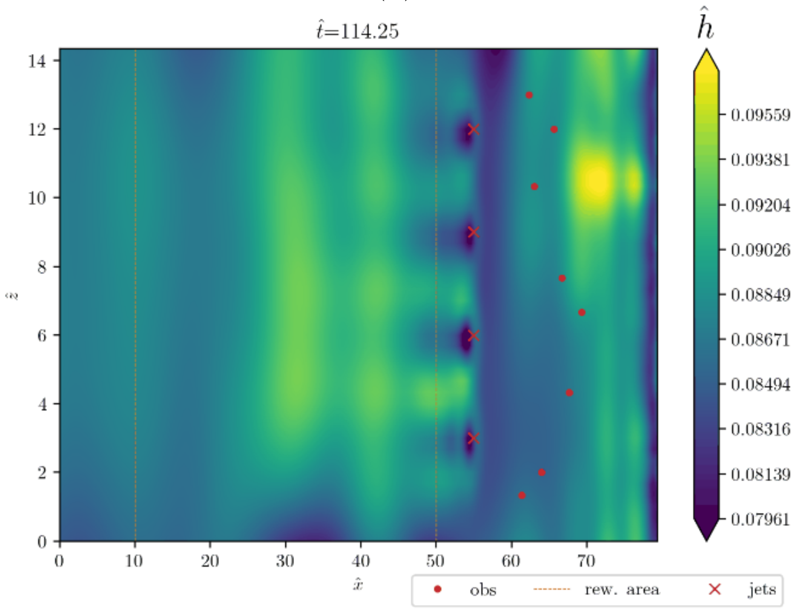
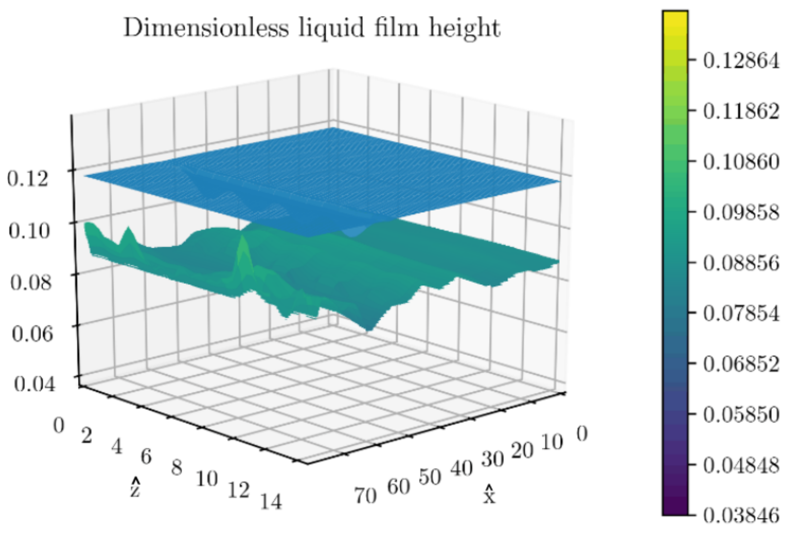


Figure 6.4: *GIF 100 ep PPO2*
Two frames of the gif as results of the DDPG.

(a)



(b)

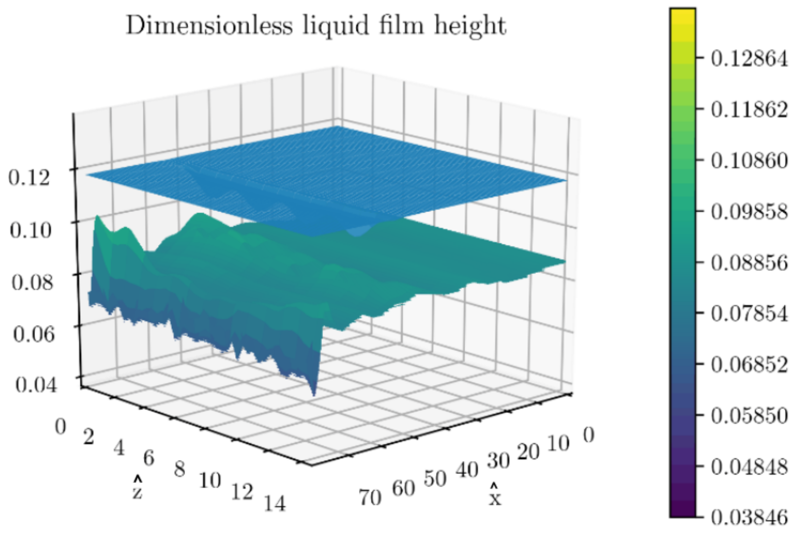


Figure 6.5: *3D GIF 100 ep PPO2 two frames of the 3D visualization*

But on the other hand, despite these results can be interesting, 100 ep are such a very low number of episodes to train well an agent. It means that the algorithm is doing an exploration phase, so it is trying several control low to find the best. For this reason, should be interesting to observe simulations with more episodes.

In this case, if we examine the behaviors of the jets in the evaluation episodes, we can observe that they predominantly remain constant (Figure 6.6). This can be attributed primarily to the normalization and definition of the action space boundaries, as well as the limited number of episodes. Essentially, the algorithm suggests that the optimal solution is to wipe more. The action space box is between [-1,1] as defined in Chapter 5 and represented in Figure 6.6, so these values are firstly normalized between 0 and 1 and then multiplied for the maximum exit flow velocity to obtain the instantaneous value of U_j. This calculated value is then provided as input to the solver to evaluate the shear stress and pressure distribution.

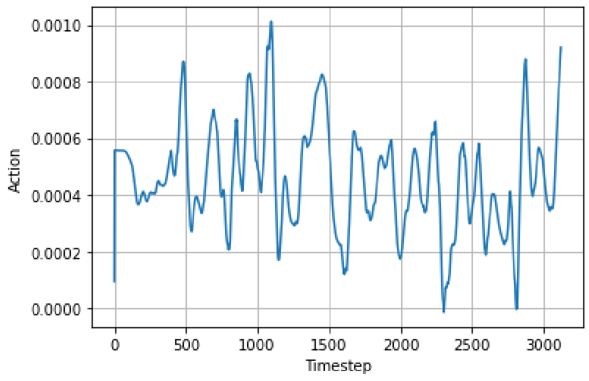


Figure 6.6: *Actions of an evaluation episode for the first jet*
These actions are in the space box [-1,1], so after normalization, they correspond to about 0.5*; it is clear that they are mainly constant because their interval is so restricted so it minces that in this case, the solution is to do an extra wiping*

Finally, let's compute the time of each simulation to evaluate the computational cost (Table 6.1).

Table 6.1: Time required for each simulation

Algorithm	Time (min)
BO	632
LIPO	1082
PPO2	763
DDPG	2913

6.1.2 Learning on 1000 episodes

Let's analyze some results in terms of learning curves with 1000 episodes (Figure 6.7)

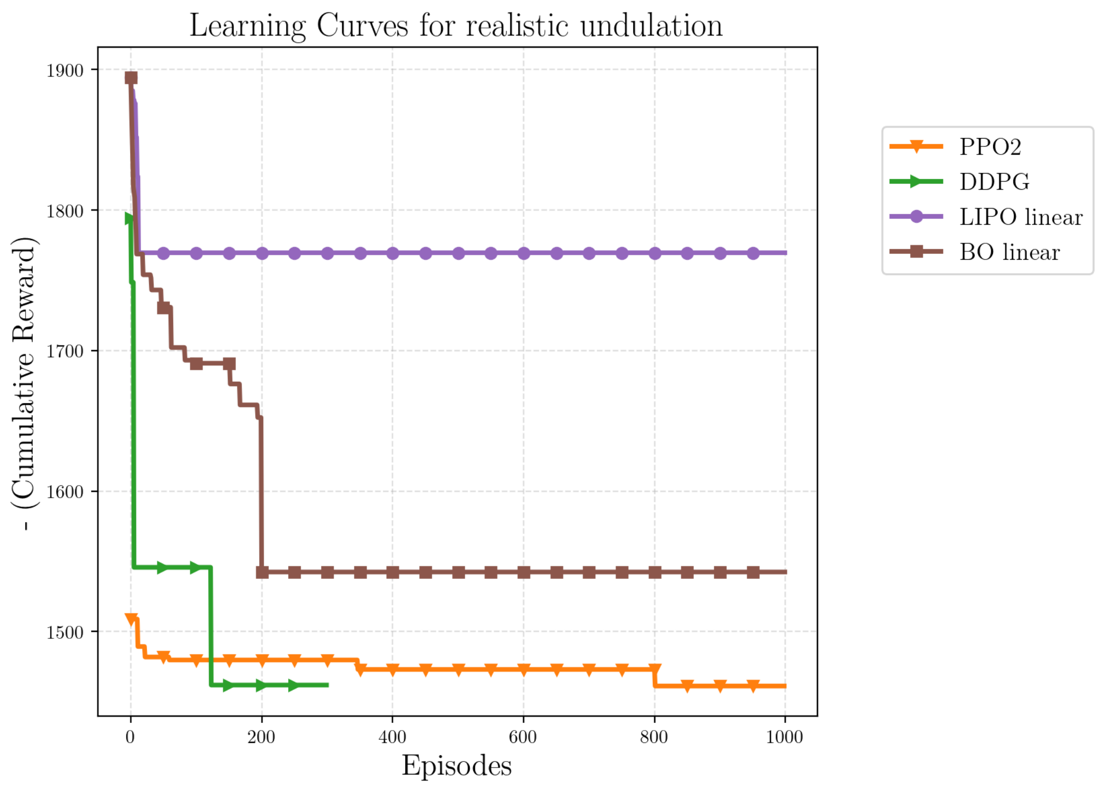


Figure 6.7: *LC 1000 ep*
After 1000 episodes of learning, we have a great improvement with PPO2

The DDPG is trained on just 300 for its high computational cost. In this case, it is possible to see that the best results are from PPO2, after HPO. The reward in this case is

$$R = -1437$$

which corresponds to about

$$25\%$$

of reduction of instabilities, which may be an interesting result in terms of final performances.

To evaluate if the policy has learned, now let's try using the model trained to test in an evaluation episode. In this case, with the PPO2, the result is an improvement of the reward compared with the uncontrolled condition, as we expected. Let's see the gif in Figure 6.8

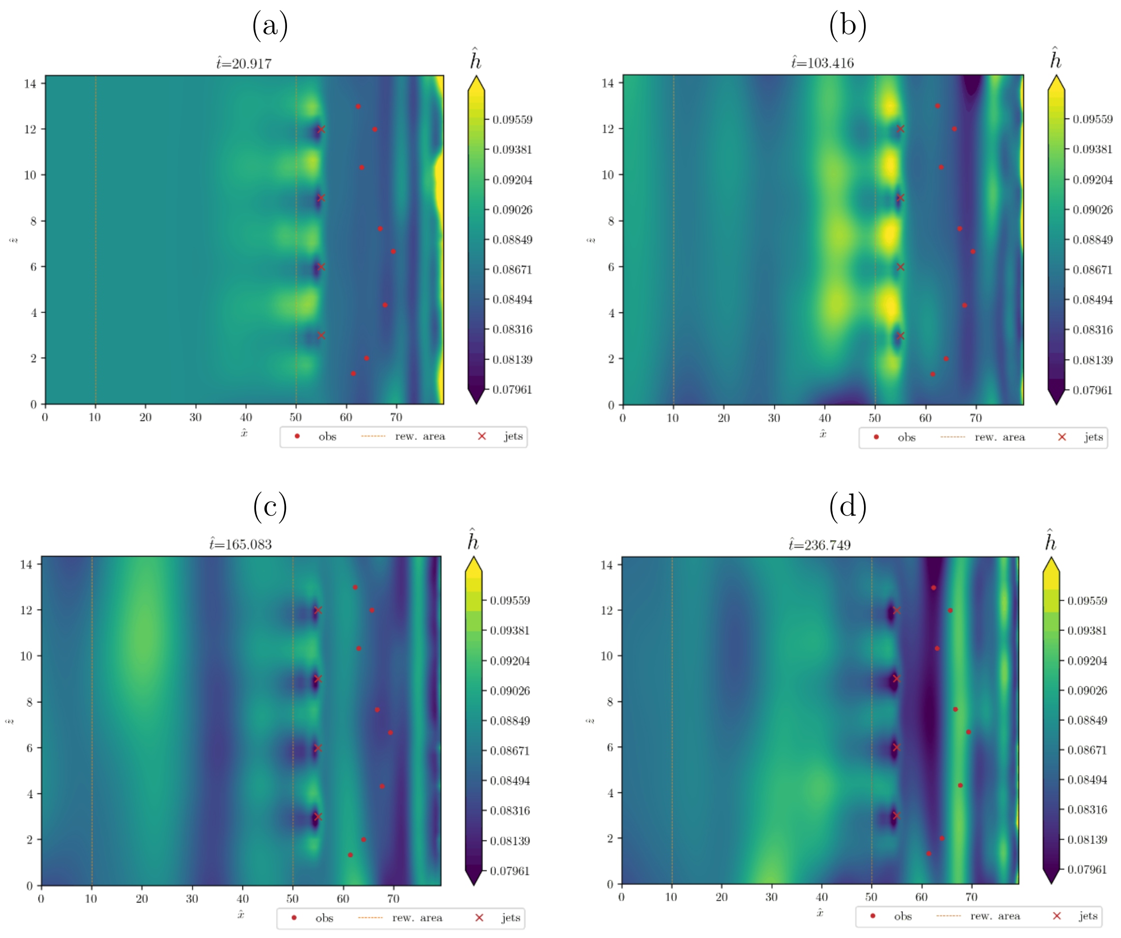


Figure 6.8: *GIF 1000 ep PPO2*
4 frames of the GIF that show the control of the thickness of liquid film

And plotting the actions of the jets it is possible to see that they are less constant than the simulations done with 100 episodes as shown Figure 6.9.

Particularly interesting is the DDPG gif (Figure 6.10),where it is possible to see that it has started to do a wave's control despite it being trained with a very low number of episodes. On the other hand, the rewards which come from the evaluation episodes are higher that the uncontrolled condition

And in this case, it can be useful to plot the actor and the critic's loss to see how good is going the learning (Figure 6.11). Initially, the actor explores various

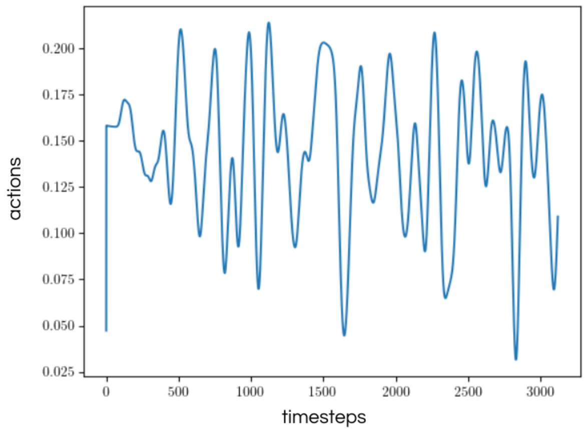


Figure 6.9: *Actions of one jet for PPO2 after learning*
After episodes learning the actions are no constant. In fact, from this plot, it is clear that the output value of ML algorithm is between 0.025 and 0.2.

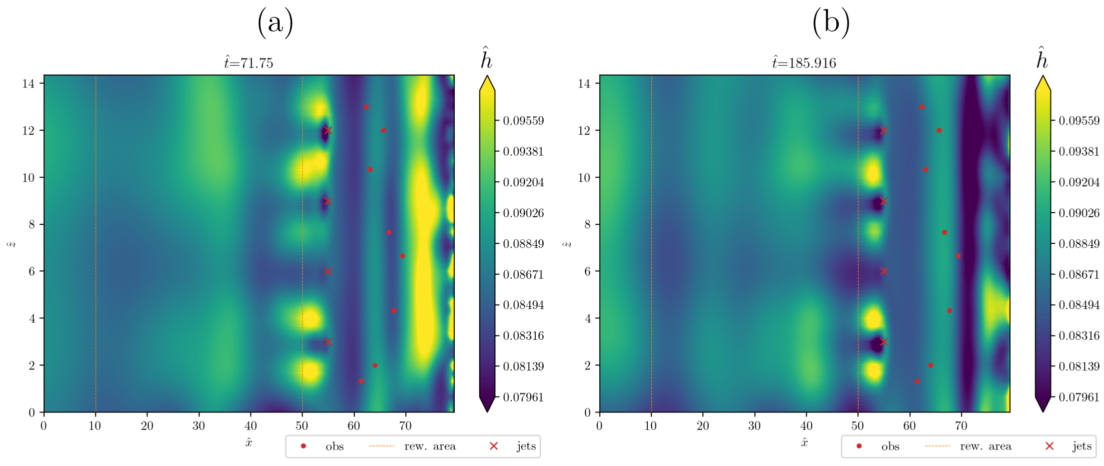
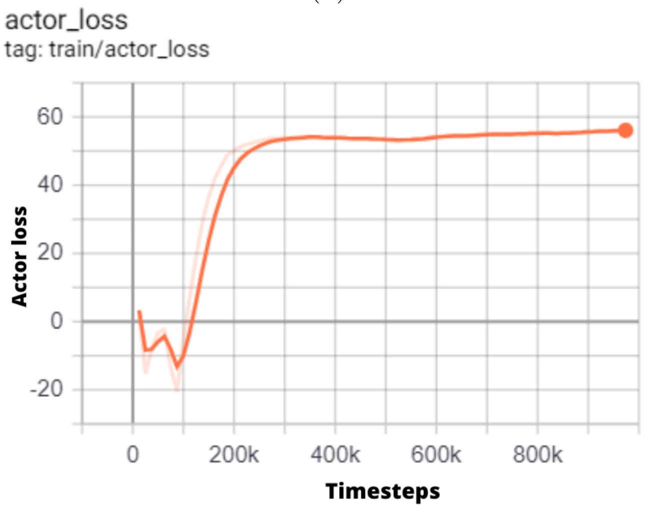


Figure 6.10: *2D GIF 300 ep DDPG*
DDPG gif, the most interesting for the control but with the agent learned with just 300 episodes for the high computational cost

actions to gather experience. As training progresses, the actor starts exploiting the learned policy, focusing on actions that have yielded higher rewards. This shift from exploration to exploitation can stabilize the loss, as the actor becomes more deterministic and focused on the most optimal actions. The convergence of the actor loss may also indicate that the algorithm has found a near-optimal policy for the given task. The actor loss measures the discrepancy between the predicted actions and the actions that maximize the expected rewards. As the algorithm learns and refines its policy, it gradually reduces this discrepancy until it reaches a point where further improvements are minimal.

(a)



(b)

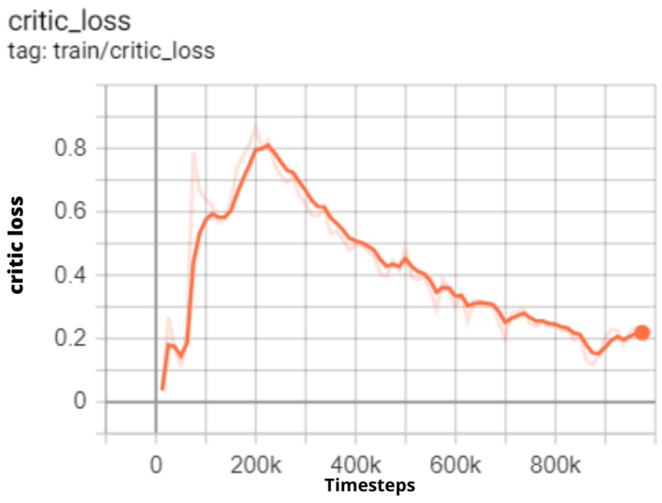
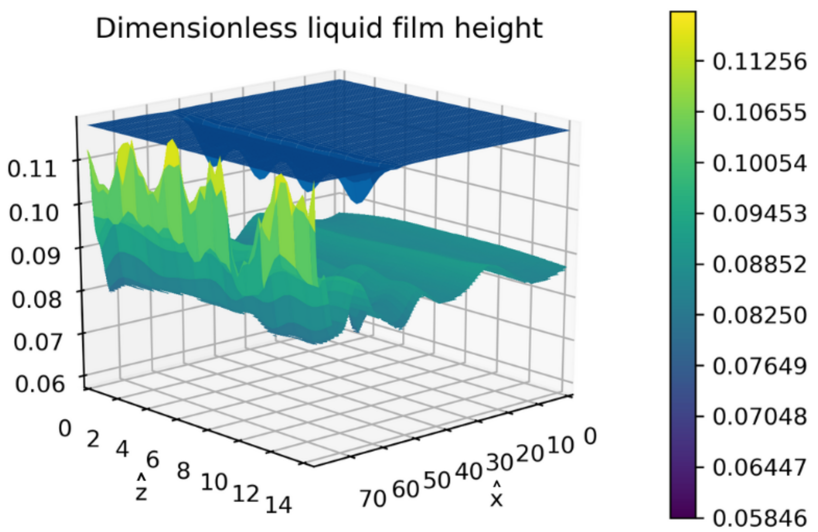


Figure 6.11: *Actor and critic loss for 300 ep (about 800000 timesteps) Actor(a) and critic(b) loss. The critic's performance is good because it goes close to zero, while the actor converges to a certain value*

Finally, the results can be represented by a 3D gif as done before in Figure 6.12

(a)



(b)

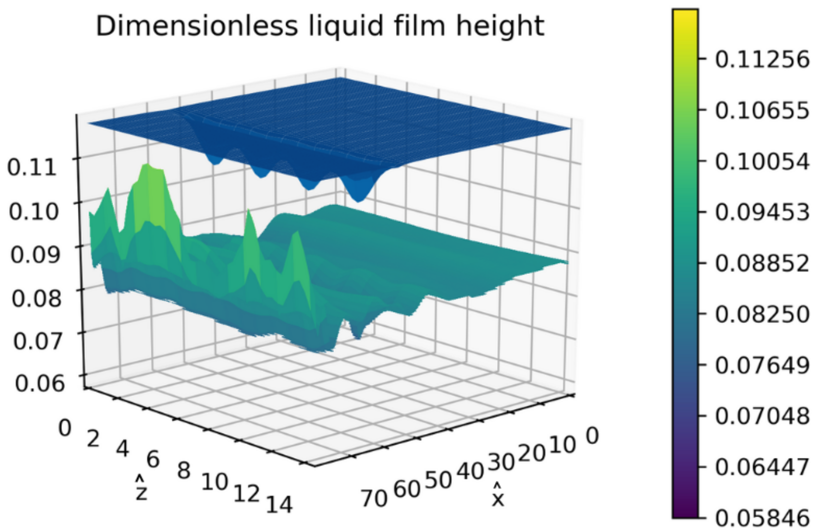


Figure 6.12: *3D GIF 300 ep DDPG two frames of DDPG 3D gif, it is clear that the jets positioned at $\hat{x}$ try to reduce the instabilities*

6.1.3 10000 episodes multiprocess environment

In this section, we discuss about the multiprocess environment, as said in Chapter 4. In this case, we ran several simulations in parallel in each processor to have more episodes for the agent's learning.

To do it, the environment should be firstly vectorized, and after that, it should be modified in a way that each processor starts from a different point with a different noise, and in this way, the exploration efficiency increase because it allows the agent to explore different areas of the environment simultaneously, potentially discovering optimal strategies faster. But on the other hand, randomizing starting points can lead to episodes with varying lengths and trajectories, resulting in more diverse training data. However, this diversity may introduce correlation issues in the training data. Correlated samples can bias the learning process and make it more challenging for the RL agent to generalize effectively. For this reason, it can be possible that the multiprocess random environment needs more episodes to learn that the serial.

Using this kind of environment it is possible to run simulations for more episodes in a much more feasible time, and it can do mainly for the reinforcement learning algorithm.

Let's see the results for the multiprocess environment with the PPO2 for about 10000 episodes as shown in Figure 6.13. It is clear that there is a descendent trend after 20 million timesteps, about 6500 episodes, and it minces that after the exploration the policy has started to learn.

After this phase, to see if the policy has been learned, let's try to do an evaluation episode using the model trained, and as we expected the result is a final reward higher than the uncontrolled condition of about

$$R = -1527$$

that is a mean of 10 episodes run using the trained agent. Figure 6.14 it is shown the reward on 10 episodes in control and uncontrolled condition with the mean. It is clear that there is a gain, but also using a multiprocess environment is very hard to go over 25% of instabilities' reduction.

After an extensive training period of 10.000 episodes, we have determined that the behaviors remain predominantly consistent during the evaluation episodes conducted with the trained model. This indicates that one of the most effective approaches is to physically engage in an extra wiping by activating a button that initiates a wiping action, as shown in Figure 6.15. It represents also a random environment, as it is possible to see from the simulation time. It means that after several episodes of learning the main behavior is to go in the direction of extra wiping and constant actions.

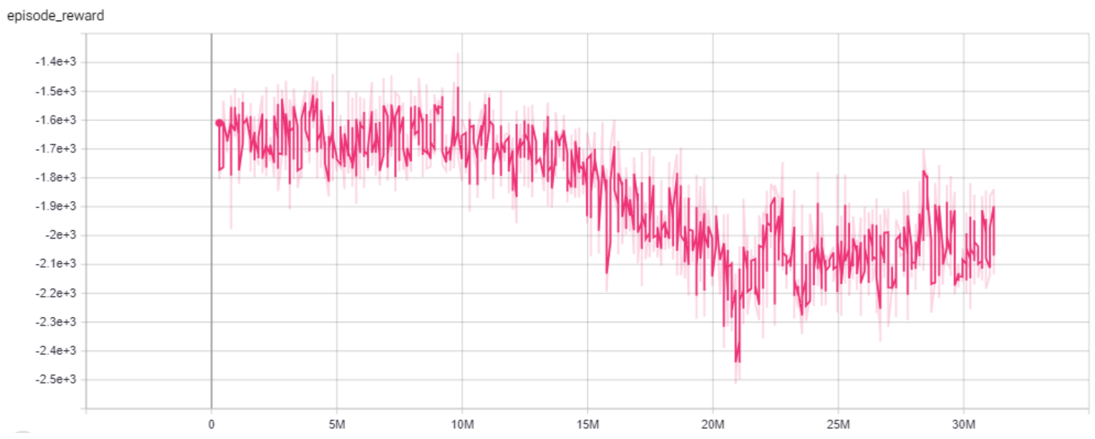


Figure 6.13: *Episode reward for 30 millions of timesteps (10k episodes)*
Episode rewards for the learning with tensorboard visualizaion

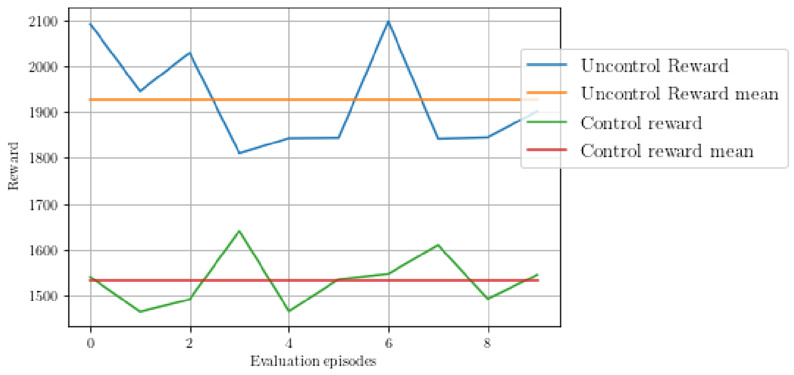
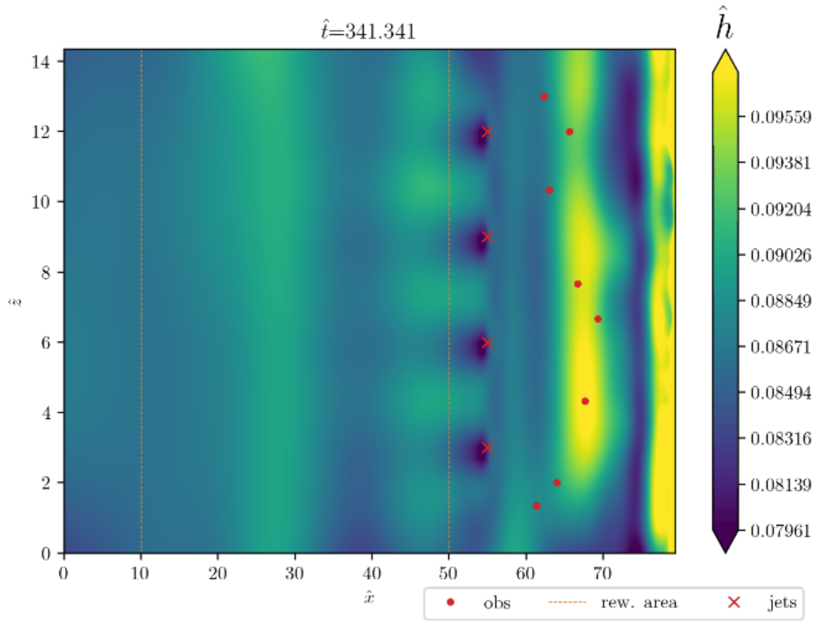


Figure 6.14: *Reward comparison on 10 evaluation episodes*
It is clear that the average reward in the controlled condition is better than the uncontrolled with a gain of about 20%

(a)



(b)

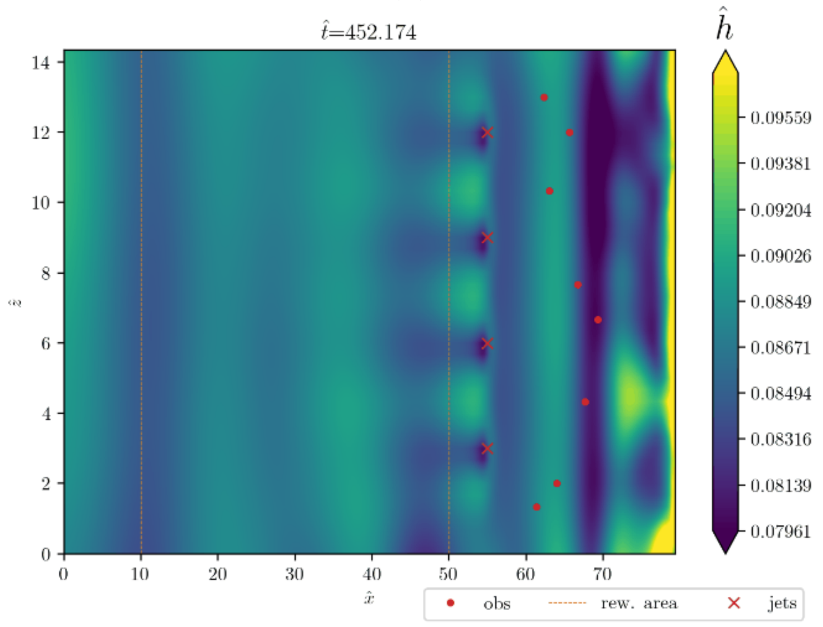


Figure 6.15: *GIF of the PPO2 with 10000 episodes of learning two frames generated using the PPO2 algorithm by a trained agent while operating within a randomized environment during evaluation episodes. The agent's actions appear to remain mainly constant.*

Conclusions

This approach demonstrates the power of data-driven techniques in tackling complex fluid dynamics problems. By leveraging the capabilities of machine learning, it has the potential to significantly enhance our understanding and control of liquid film instabilities.

From some simulations, after learning about the agent, it is clear that one of the possibilities to reduce undulation is to give constant actions. Physically this solution means that one way to reduce waves and instabilities is to do an extra wiping. The key finding highlights the intricate nature of this control problem, suggesting that additional wiping action after the first phase of thickness regulation is highly effective also for undulation reduction. Consequently, for such applications, employing an open-loop control mechanism appears to be more advantageous than a closed-loop control. This assertion is supported by numerous simulations where the actions remain predominantly consistent, in particular after the PPO2 with 10000 episodes of learning. This analysis can be useful to define the parameters of the open-loop control, such as the exit flow velocity.

After the optimization of all the main features of the RL algorithm and custom environment, we arrive to reduce the reward by about 25%, which can be an interesting result that should be tested. Of course, if the number of episodes of learning increases the agent can learn better, and it is possible to further reduce the instabilities.

Comparing several algorithms for the control, we can conclude that the PPO2 is the most efficient in terms of final reward. On the other hand, DDPG exhibits superior learning capabilities and offers intriguing alternatives due to its resemblance to closed-loop control. However, its extensive computational requirements pose challenges when attempting to train it with a larger number of episodes.

In conclusion, by utilizing an ML algorithm to reduce the instabilities of a 3D liquid film, it is possible to achieve a more efficient coating or deposition process with a considerable reduction of undulation. By automating the control of the coating process, ML algorithms can make real-time adjustments to achieve the desired thickness more efficiently, saving both time and resources.

Future trends

This thesis should be a starting point for other successful research. After a preliminary analysis, we define a final setup that can be further improved for sure.

- Firstly, we should try to run this simulation with more episodes, e.g. 100000 for the serial DDPG, to evaluate the learning performance of the agent. These simulations are very computationally intensive, so a good strategy for the future is to run them on more powerful computers or vectorize the environment to run the same environment on more computers in parallel. Another issue is to find a new and more complex cost function that may be more efficient.

- It should be interesting to compare several integral models presented in [2], such as the ILB and the WILB to see the difference. Another analysis can be done by comparing the results obtained with integral models and the solutions of NS equations using a DNS.

- Robustness analysis might also be of interest to also evaluate the impact of noise and give the algorithm more reliability to be tested in the real world.

- Another idea is to use new relations for pressure and shear stress distribution: those of Beltaos, are such an old experimental and empirical law that it should be interesting to find new equations. If we have the opportunity to conduct fresh experiments, it would certainly be intriguing to explore novel hypotheses by examining alternative scenarios. For instance, by introducing a specific variable, let's say alpha as a nozzle angle and incorporating an additional degree of freedom, we can strive to discover enhanced efficiencies or uncover previously unknown laws.

- The world of ML is getting better every day, so another point is to test this environment with other ML algorithms using new versions released by stable baselines 3, such as Twin Delayed DDPG (TD3) and Soft Actor-Critic (SAC), which are similar to DDPG, and Sample Efficient Actor-Critic with Experience Replay (ACER). In complex liquid film control scenarios, multiple agents can collaborate or compete to optimize different aspects of the system. Using multi-agent reinforcement learning techniques can enable coordinated control actions and handle complex interactions between liquid film regions or different control objectives. By continuously updating and retraining the models with new data, the algorithms can refine their predictions and control strategies. This adaptability allows for better precision and responsiveness in maintaining the desired liquid thickness.

- As machine learning techniques become more prevalent in real-world applications, the need for explainability and interpretability increases. Researchers may focus on developing methods to interpret the decisions made by machine learning models in liquid film thickness control systems. Explainable AI techniques can provide insights into why certain control actions are taken, increasing trust and enabling easier integration with existing control frameworks.

- On the other hand, despite machine learning offers promising results for liquid thickness control, it's important to acknowledge potential challenges. Data quality, availability, and representativeness are crucial factors for training accurate models. The need for continuous monitoring, model updates, and potential system complexities may also pose implementation challenges.

- Finally, in a few years, it will be interesting to test the control in the plant that will be built in Canada to get real feedback and compare the numerical results with the experimental ones. In this case, may be interesting to integrate machine learning techniques with traditional control methods that can combine the strengths of both approaches. Combining model-based control strategies with data-driven approaches can provide better stability guarantees and handle scenarios with limited data or challenging dynamics.

Bibliography

[1] Anne Gosset, Miguel Alfonso Mendez, and Jean-Marie Buchlin. «An experimental analysis of the stability of the jet wiping process: Part i–characterization of the coating uniformity». In: *Experimental Thermal and Fluid Science* 103 (2019), pp. 51–65 (cit. on pp. 3, 23).

[2] Miguel Alfonso Mendez, Anne Gosset, Benoıt Scheid, Mikhael Balabane, and Jean-Marie Buchlin. «Dynamics of the jet wiping process via integral models». In: *Journal of Fluid Mechanics* 911 (2021), A47 (cit. on pp. 4–6, 67).

[3] *Skopt*. `https://scikit-optimize.github.io/stable/modules/generated/skopt.Optimizer.html` (cit. on p. 9).

[4] Ashley Hill et al. *Stable Baselines*. `https://github.com/hill-a/stable-baselines`. 2018 (cit. on p. 9).

[5] *tensorflow 1.14*. `https://www.tensorflow.org/`. 2021 (cit. on p. 9).

[6] Fabio Pino, Lorenzo Schena, Jean Rabault, and Miguel Alfonso Mendez. «Comparative analysis of machine learning methods for active flow control». In: *Journal of Fluid Mechanics* 958 (2023), A39 (cit. on pp. 9, 16).

[7] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. «Taking the human out of the loop: A review of Bayesian optimization». In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175 (cit. on pp. 12, 13).

[8] Cédric Malherbe and Nicolas Vayatis. «Global optimization of Lipschitz functions». In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2314–2323 (cit. on pp. 14, 15).

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. «Proximal policy optimization algorithms». In: *arXiv preprint arXiv:1707.06347* (2017) (cit. on pp. 14, 15).

[10] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. «Continuous control with deep reinforcement learning». In: *arXiv preprint arXiv:1509.02971* (2015) (cit. on pp. 16, 17).

[11] *OpenAI gym.* `https://www.gymlibrary.dev/` (cit. on p. 17).

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym.* 2016. eprint: `arXiv:1606.01540` (cit. on p. 17).

[13] Spyridon Beltaos and Nallamuthu Rajaratnam. «Impinging circular turbulent jets». In: *Journal of the hydraulics division* 100.10 (1974), pp. 1313–1328 (cit. on pp. 22, 24, 25).

[14] *PyDOE.* `https://pythonhosted.org/pyDOE/` (cit. on p. 30).

[15] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. «Optuna: A Next-generation Hyperparameter Optimization Framework». In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 2019 (cit. on p. 34).

[16] *Neural Network Architectures.* `https://freecontent.manning.com/neural-network-architectures/` (cit. on p. 38).