Student Thesis No. 837

Robust model-based deep reinforcement learning for flow control

Janis Geise

Examiner: Prof. Dr.-Ing. Rolf Radespiel
Institute of Fluid Mechanics
Head of Institute: Prof. Dr.-Ing. R. Radespiel
Braunschweig University of Technology

Supervisor: Dr.-Ing Andre Weiner (TU Braunschweig)

Publication: January 2023

Affidavit

I, Janis Geise, declare that I have authored this student thesis independently, that I have not used other than the declared sources and resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Braunschweig, 31.01.2023

Abstract

Active flow control has the potential of achieving remarkable drag reductions in applications for fluid mechanics, when combined with deep reinforcement learning (DRL). The high computational demands for CFD simulations currently limits the applicability of DRL to rather simple cases, such as the flow past a cylinder, as a consequence of the large amount of simulations which have to be carried out throughout the training. One possible approach of reducing the computational requirements is to substitute the simulations partially with models, e.g. deep neural networks; however, model uncertainties and error propagation may lead an unstable training and deteriorated performance compared to the model-free counterpart. The present thesis aims to modify the model-free training routine for controlling the flow past a cylinder towards a model-based one. Therefore, the policy training alternates between the CFD environment and environment models, which are trained successively over the course of the policy optimization. In order to reduce uncertainties and consequently improve the prediction accuracy, the CFD environment is represented by two model-ensembles responsible for predicting the states and lift force as well as the aerodynamic drag, respectively. It could have been shown that this approach is able to yield a comparable performance to the model-free training routine at a Reynolds number of $Re = 100$ while reducing the overall runtime by up to 68.91%. The model-based training, however, showed a high dependency of the performance and stability on the initialization, which needs to be investigated further. An increase of the Reynolds number to $Re = 500$ and $Re = 1000$ revealed several issues within the model-free training routine, such as a dependency of the stability of the policy optimization on its initialization, which were encountered in the subsequently conducted model-based trainings as well.

Contents

Nomenclature **vii**

1 Introduction **1**
1.1 Related work . . . 1
1.2 Present work . . . 4

2 Theoretical background **5**
2.1 Reinforcement learning . . . 5
2.1.1 General principle of reinforcement learning . . . 5
2.1.2 Markov decision process . . . 5
2.1.3 Deep reinforcement learning . . . 6
2.2 Model-free DRL . . . 6
2.2.1 Proximal policy optimization . . . 6
2.2.2 Disadvantages of model-free DRL algorithms . . . 7
2.3 Model-based DRL . . . 8
2.3.1 Challenges of model-based DRL . . . 8

3 Numerical method **10**
3.1 Flow past a cylinder . . . 10
3.1.1 Numerical setup of the flow problem . . . 10
3.1.2 Domain discretization . . . 11
3.1.3 Adjustments for higher Reynolds numbers . . . 13
3.2 Libraries and computational resources . . . 14
3.2.1 Phoenix HPC . . . 14
3.2.2 AWS . . . 14
3.2.3 PyTorch . . . 14
3.2.4 OpenFOAM . . . 14
3.2.5 Apptainer . . . 15
3.2.6 OpenMPI . . . 15
3.2.7 Drlfoam . . . 15

4 Modeling the CFD environment **16**
4.1 Influence of the buffer size and trajectory length for MF-training . . . 16
4.2 Overview of possible approaches for MB-DRL . . . 20
4.2.1 One global environment model vs. one new model each episode . . . 21
4.2.2 Influence of the model architecture . . . 24
4.2.3 Separating predictions of c_L and p_i from predicting c_D . . . 25
4.2.4 Influence of the number of input time steps on the prediction error . . . 27
4.2.5 Low-pass filtering of the c_D trajectories . . . 30
4.2.6 Prediction of the change of state instead of the next state . . . 30

5 Results **32**
5.1 Initial approach of integrating a MB-training into *drlfoam* . . . 32
5.1.1 Optimizing the number of episodes for training the environment models . . . 33
5.1.2 Influence of the trajectory length . . . 35
5.1.3 Assessment of the final policies . . . 37
5.1.4 Extension to model-ensemble . . . 39
5.1.5 Optimization of the training routine . . . 43
5.1.6 Conducting a MB-training on different systems . . . 44

5.2 Generalization of the training routine . . . 46
5.2.1 Early stopping of the model-training . . . 46
5.2.2 Comparison of the prediction accuracy . . . 47
5.2.3 Influence of the number of models within the ensemble . . . 49
5.2.4 Alternation between model-based and model-free episodes . . . 51
5.2.5 Improvements of the training routine and migrating to AWS . . . 54
5.3 Comparison of the final results for $Re = 100$ . . . 59
5.4 Model-based training at higher Reynolds numbers . . . 61
5.5 Comparison of the final results for $Re = 500$ . . . 64

6 Conclusion **65**

Bibliography **68**

List of Figures **71**

List of Tables **74**

A Appendix **75**
A.1 Navier-Stokes equations for incompressible flow . . . 75
A.2 Adjustments for higher Reynolds numbers . . . 76
A.3 Influence of the buffer size and trajectory length (MF-DRL) . . . 77
A.4 Integration of the model-based approach into *drlfoam* . . . 78
A.5 Hyperparameter settings . . . 80
A.5.1 PPO . . . 80
A.5.2 One global environment model vs. one new model each episodes . . . 80
A.5.3 ME-MB-DRL . . . 80
A.6 Generalization of the training routine . . . 81
A.6.1 Network architecture study new training routine . . . 81
A.6.2 Final improvements to the new training routine . . . 83
A.7 Final results for a Reynolds number of $Re = 100$ . . . 85
A.8 Model-based training at higher Reynolds numbers . . . 86

Nomenclature

Latin symbols

A	action space	$[-]$
a	actions	$[-]$
b	buffer size	$[-]$
c_D	drag coefficient	$[-]$
c_D^*	drag coefficient scaled to interval $[0, 1]$	$[-]$
c_L	lift coefficient	$[-]$
c_L^*	lift coefficient scaled to interval $[0, 1]$	$[-]$
d	diameter	$[m]$
l	trajectory length	$[s]$
l^*	dimensionless trajectory length	$[-]$
p	static pressure	$[N/m^2]$
R	correlation coefficiant	$[-]$
r	radius, rewards	$[m, -]$
r^*	rewards scaled to interval $[0, 1]$	$[m]$
Re	Reynolds number	$[-]$
R	reward space	$[-]$
S	state space	$[-]$
s	states	$[-]$
t	time	$[s]$
t^*	time scaled to interval $[0, 1]$	$[-]$
u	velocity of the free stream in x-direction	$[m/s]$
$\boldsymbol{u}$	velocity vector	$[m/s]$
$\boldsymbol{u^*}$	dimensionless velocity vector	$[-]$
v	velocity of the free stream in y-direction	$[m/s]$
$\mathcal{W}$	entropy	$[-]$
x	x-component in cartesian coordinate system	$[m]$
x^*	dimensionless x-component in cartesian coordinate system	$[-]$
$\boldsymbol{x}$	coordinate vector in cartesian coordinate system	$[-]$
$\boldsymbol{x^*}$	dimensionless coordinate vector in cartesian coordinate system	$[-]$
y	y-component in cartesian coordinate system	$[m]$
y^*	dimensionless y-component in cartesian coordinate system	$[-]$

Greek symbols

α	first parameter of beta-distribution	$[-]$

β	second parameter of beta-distribution	$[-]$
Δ	Laplace operator	$[-]$
γ	discount factor	$[-]$
μ	mean	$[-]$
σ	standard deviation	$[-]$
∇	Nabla operator	$[-]$
ν	kinematic viscosity	$[Pa * s]$
ω	rotational velocity of the cylinder	$[1/s]$
π	policy	$[-]$
π^*	optimal policy	$[-]$
ρ	density	$[kg/m^3]$

Abbreviations

AR	Aspect ratio
AWS	Amazon web services
CFD	Computational fluid dynamics
CFL	Courant-Friedrichs-Lewy number
CPU	Central processing unit
DRL	Deep reinforcement learning
GAE	Generalized advantage estimation
GPU	Graphics processing unit
MDP	Markov decision process
HPC	High performance computing
MB	Model-based
ME	Model ensemble
MF	Model-free
ML	Machine learning
MSE	Mean squared error
NN	Neural network
PSD	Power spectral density

Indices

∞	free stream	
Θ	parameters of a neural network	$[-]$

Chapter 1

Introduction

Increasing fuel cost and environmental concerns are driving the necessity of reducing CO_2 and NO_x emissions in aviation. Since an electrification of aircraft is significantly more challenging than e.g. for cars, other possibilities for the reduction of fuel consumption need to be investigated. One promising approach is referred to as flow control, aiming to reduce for example the drag of an airfoil. Flow control methods can hereby divided into passive flow control (PFC) and active flow control (AFC) [2]. Drag reduction using passive flow control methods can be achieved by an optimization of the wing surface or the airfoil geometry, e.g. in order to increase the amount of laminar flow, which is generally known as natural laminar flow (NLF) [15]. However, passive flow control methods need to be optimized for a certain design point limiting the possibilities with respect to the drag reduction and capabilities for generalization. Active flow control methods on the other hand use additional energy, e.g boundary layer suction, to actively control the flow based on the current flow conditions [2]. This method, although requiring additional energy, shows a greater potential with respect to the achievable drag reduction while mitigating the disadvantages of passive flow control methods. Finding an optimal control law, however, is not trivial since the required amount of e.g. suction is not only dependent on the current design point but influenced by outer disturbances such as wind gusts as well, making a manual definition of an optimal control law not feasible. The developments in artificial intelligence (AI) and especially deep reinforcement learning (DRL) over the last years show great capabilities of learning optimal control laws in high dimensional action spaces [50]. DRL is already utilized in fluid mechanics e.g. for turbulence modeling [28, 16, 23] or solving partial differential equations [34]. Combining DRL with computational fluid dynamics (CFD) in order to solve flow control problems yielded promising results so far, but the usage of CFD and associated requirements with respect to the computational resources limits the applicability of DRL for active flow control currently to rather simple problems.

1.1 Related work

Although there have been recent studies which utilize DRL for more complex flow control problems, e.g. in [10], the two-dimensional flow past a cylinder is a frequently used test case for the application of active flow control using DRL. This is reasoned by the simplicity of the numerical setup and well-known flow characteristics. The flow around a cylinder was thoroughly benchmarked by [36], providing the definition of the numerical domain, inlet and boundary conditions as well a characteristic values for c_L and c_D. As a consequence, the results of this study were later used in a variety of subsequent studies focusing on the application of DRL for active flow control.

Controlling the flow past a cylinder using DRL was successfully conducted firstly by [31] at a Reynolds number of $Re = 100$. In order to reduce the lift and drag acting on the cylinder, an

injection of mass flow at the cylinder surface was applied. The agent was trained with the PPO-algorithm [40] and finally able to achieve a drag reduction by 8% while only injecting a mass flow rate of 0.5% of the total mass flow through the domain. However, this approach yielded high runtimes since the agent only interacted with a single environment each episode limiting the ability of testing more challenging environments. It was found by [32], that the execution time of the CFD simulations made up 99.7% of the complete training time, which motivated to improve the current implementation further. In order to decrease the computational demands and runtime, this approach was optimized in a next step by [32]. The execution of the CFD simulation was parallelized, which enabled the agent to learn with the experiences of multiple environments at the same time. With this approach, it was possible to decrease the required runtime by a factor of up to 60 compared to a training with a single environment [32].

An alternative approach of active flow control was taken by [8], here the A2C-algorithm was utilized in order to train the agent. The agent learned to control the flow by actuating two plasma actuators with a certain burst frequency instead of two continuous jets as conducted in [32]. The bursts lead to a reattachment of the flow and as a consequence were able achieve a drag reduction of 22.6%.

In a next step, [26] modified the PPO-algorithm aiming to optimize the sensor layout of the probes placed in the flow field. The agent was trained for flows with Reynolds numbers between $Re = 100...216$. At a Reynolds number of $Re = 120$, a drag reduction of 18.4% could have been achieved by using 12 pressure probes placed in the wake of the cylinder. However, it was shown that the number of required pressure probes can be decreased to five without a experiencing a significant deterioration with respect to the achieved performance.

A different approach for training the agent with PPO was investigated by [30]. Here, the setup of [36] was used as well, but instead of defining the reward function with the obtained c_L and c_D values, here dynamic mode decomposition (DMD) is utilized. Further, a direct numerical simulation (DNS) was used in order to provide high-fidelity data of the flow field, the flow field is then decomposed using DMD at the 63 probes placed in the wake of the cylinder. Overall, a drag reduction of 8% could have been achieved, which is in accordance to the results presented in [31].

The numerical setup of [31] was adapted by [45] in order to analyze the ability for generalization of the training routine with respect to the Reynolds number. Therefore, the number of jets responsible for the mass flow injection was increased to four instead of two jets. The PPO-algorithm was utilized to train the agent on flows with Reynolds numbers of $Re = 100, 200, 300$ and $Re = 400$. The agent was finally able to achieve drag reductions between 5.7% for $Re = 100$ and 38.7% for $Re = 400$. Further, this algorithm showed a great potential of adopting to flows with different Reynolds numbers and therefore able to yield a good performance on any unseen Reynolds number between $Re = 60...400$ [45].

The next large improvement to the ability of controlling the flow past a cylinder was made by [49]. Here, stable trainings with PPO up to a Reynolds number of $Re = 2000$ using PPO and two jets at the cylinder surface were conducted. The chosen setup was hereby the same as the one used in [31], the bursts suggested by the agent were smoothed in order to achieve a continuous mass flow rate with respect to the time analogously to the work done by [32]. The agent was able to reduce the drag by 17% for $Re = 2000$ while a decrease of even 19% was achieved at $Re = 1000$. It further could have been observed that in contrast to low Reynolds numbers, the agent increases the frequency of the actuation significantly in order to delay the detachment of the flow by forcing a breakdown of the large vertices into smaller ones [49]. Additionally, it was tested if the training time of the agent can be reduced if the training is not initialized with a random policy but with a pre-trained policy of trainings with lower Reynolds number. It was found that this approach yields good results if the flow physics of these two cases are similar.

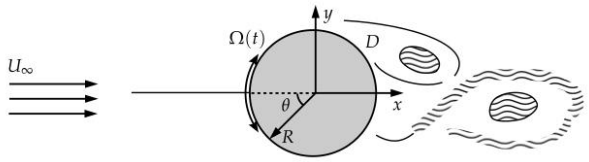


Figure 1.1: Rotation of the cylinder during the flow simulation [47]

However, if the differences within the Reynolds number become too large, e.g. initializing a training for $Re = 2000$ with a policy trained with flows at $Re = 100$, the training yields no improvements with respect to the ability of controlling the flow.

Although the agent was able to efficiently control the flow in all presented studies, the overall approach of using an injection of mass flow, either continuous or by discrete bursts, to delay the detachment of the boundary layer remained the same across all studies. A different approach for the actuation was developed by [47], here instead of a mass flow injection, the cylinder was rotated around the $z-$axis with a variable angular velocity $\Omega(t)$ as depicted in fig. 1.1. The goal of the agent was to find the optimal control law, meaning the angular velocity depending on the current state of the flow simulation. This approach was able to achieve drag reductions up to 16% for a Reynolds number of $Re = 100$ while the angular velocity required a maximum percentage of the inflow velocity of 8% and 0.8%, respectively, depending on the reward function used [47].

The approach of [47] in combination to the PPO-algorithm implemented in the work of [32] was adopted by [6] and optimized further by [9]. A qualitative comparison of the controlled flow using the implementation of [6] is depicted in fig. 1.2, here the tangential velocity of the cylinder surface corresponds to a maximum of 25% of the average inflow velocity. It can be seen clearly that the agent is able to reduce the vortex shedding in the wake of the cylinder significantly. However, the training of the agent, although a multi-environment approach is utilized requires long runtimes and high computational costs.

The work of [41] finally aimed to alter the model-free training routine implemented in [6] towards a model-based approach in order to accelerated the training of the agent. Recurrent neural networks, trained on data of the model-free training, were utilized for approximating the CFD environment, however, this approach was not able to maintain the required prediction accuracy over the course of the training.

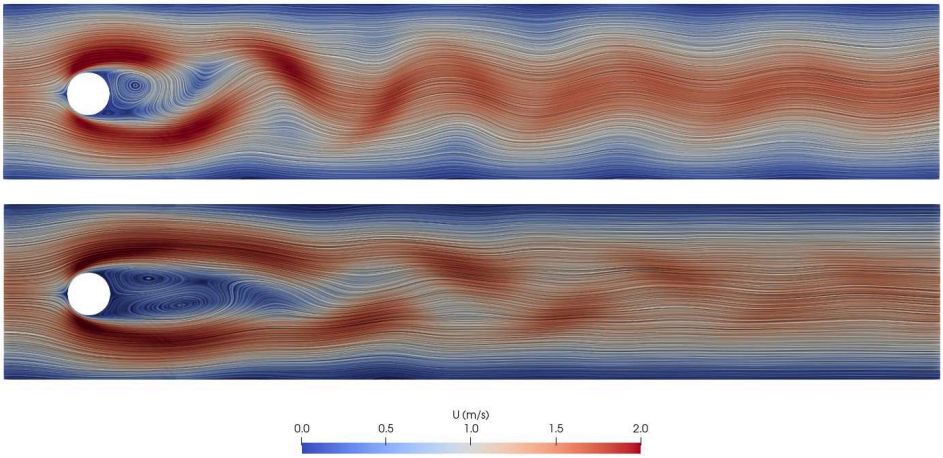


Figure 1.2: Comparison of the uncontrolled flow (top) with the controlled flow (bottom) at $Re = 100$ using the model-free DRL algorithm implemented by [6]

1.2 Present work

This project[1] is a continuation of the work done by [6] and [9], a first attempt to use a model-based approach in order to accelerate the training process was implemented by [41]. The *drlfoam* framework, which is based on the work of [6] and [9] provides the model-free training routine used as a starting point. The objective of this thesis is to derive requirements for a potential model-based approach in a first step. Therefore, the influence of the buffer size as well as the trajectory length on the training stability and performance with respect to the received rewards will be investigated. In a next step, various different approaches to model the CFD environment with fully-connected deep neural networks will be tested, these approaches operate on the generated data of the model-free PPO-training aiming to proof the general feasibility of model-based DRL for controlling the flow past a cylinder. The most promising approach is then integrated into the PPO-training routine and successively improved further. After the successful integration of the model-based approach into the PPO-training routine, the training stability with respect to the derived requirements is analyzed, a special emphasis will be payed on the influence of the trajectory length as well as the alternation between model-based and model-free episodes. Lastly, it will be investigated if the model-based approach can be adopted to flows with higher Reynolds numbers.

[1] The code developed in this thesis is publicly available under:
`https://github.com/JanisGeise/robust_MB_DRL_for_flow_control/`

Chapter 2

Theoretical background

There exits a vast variety of algorithms and methods for reinforcement learning (RL) and deep reinforcement learning (DRL) in order to solve complex, high-dimensional problems with machine learning. The applicability of such algorithms, however, requires a fundamental understanding of the underlying mathematical principles and differences within these methods and is therefore not straightforward to implement. This section will give an overview of the fundamentals of reinforcement and deep reinforcement learning, followed by an introduction into the two general approaches, namely model-free DRL (MF-DRL) and model-based DRL (MB-DRL).

2.1 Reinforcement learning

Reinforcement learning (RL) refers to the part of machine learning aiming to imitate the human learning process [7]. In contrast to other machine learning techniques such as supervised learning, in RL a controller (agent) gathers experience by interacting with an environment [44]. These interactions are utilized by the agent to develop a strategy (policy) on how solve the given task in the most efficient way. Comprehensive explanations on RL can be found e.g in [13, 3, 33] and [7].

2.1.1 General principle of reinforcement learning

Figure 2.1 depicts the general principle of RL. As mentioned, the agent interacts with an environment by taking an action a_t. The environment is hereby in a current state s_t and transitions to a next state s_{t+1} due to a taken action a_t [13]. All states s_i are part of the state space $s \in S$, denoting all possible states which can be transitioned to or from. The actions are constrained by the action space $a \in A$ denoting all possible actions which can be taken within the environment. Based on the taken action a_t, the agent receives a reward r_t from the environment along with the next state s_{t+1}. In case the agent has no access to the full state s, instead of the state it receives an observation o which is a part of the state, $O \subset S$. The reward r lies in the reward space $r \in R$, denoting all possible rewards achievable. The agent is then adjusting its policy π in order to maximize the rewards based on the received observations and the corresponding reward. The policy hereby denotes the set of rules the agent is following for sampling its actions from [3]. To accelerate the learning process, multiple interactions with the environment are collected before feeding back to the agent. This so-called trajectory may contain either a finite or infinite amount of experience tuples with the encountered states, actions and rewards [13].

2.1.2 Markov decision process

The Markov decision process (MDP) is a mathematical formalization of the aforementioned principle of RL. The transition from $s_t \rightarrow s_{t+1}$ underlies probabilistic principles rather than

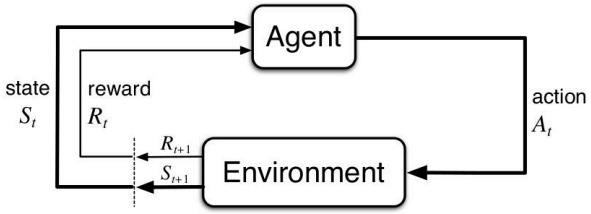


Figure 2.1: Principle of reinforcement learning [44]

being deterministic, due to the fact that the transition dynamics of the environment are generally not or only partially known by the agent [37]. Consequently, there exists a transition probability $\mathbb{P}(s_{t+1}|s_t, a_t)$ for transitioning to the next state s_{t+1} when being in a state s_t and a given a_t [35]. The action a_t is generally sampled from a probability distribution given by the policy as $\pi(a_t|s_t)$ and therefore also non-deterministic [37].

In practice multiple experiences of interactions between the environment and the agent are collected and saved as trajectory before updating the policy. The length of the trajectory is referred to as the horizon, which can either be finite or infinite depending on the given task. A discount factor γ is introduced in order to make recent events more valuable compared to events in the past as

$$R = \sum_{t=0}^{\infty} \gamma^t r_{t+1}, \qquad \gamma \leq 1 \tag{2.1}$$

where R denotes the cumulative reward of the trajectory [35]. A discount factor of $\gamma = 0$ would mean that the agent only takes the action yielding the highest reward at this time step and therefore acting greedy, without considering that another action which is unfavorable at the moment but may lead to a higher reward in the future [44].

2.1.3 Deep reinforcement learning

The state and action spaces of physical systems in general is high-dimensional, e.g. in CFD each grid point contains the flow properties of each time step while (for subsonic flow) all points in the flow field are affecting each other. A manual definition of the policy or the usage of tabular methods as it is done e.g. in Q-learning is therefore unfavorable due to the high computational costs [35]. Deep neural networks as non-linear function approximators on the other hand are great at mapping high-dimensional inputs to outputs and have therefore a large potential for approximating arbitrary policies [7]. The combination of RL and deep neural networks is referred to as deep reinforcement learning (DRL).

2.2 Model-free DRL

In model-free DRL (MF-DRL), the agent interacts directly with the environment as depicted in fig. 2.1. There have been many algorithms developed over the last years, e.g. SAC [14] or TRPO [38]. Since this thesis utilizes proximal policy optimization (PPO) [40] for training the agent, this algorithm will be presented briefly in the following. A thorough description of the implementation of PPO can be found in the previous work of [6] and [9].

2.2.1 Proximal policy optimization

Proximal policy optimization (PPO) is based on TRPO and developed by [40], but in contrast to TRPO the implementation and usage is significantly simpler. The PPO-algorithm consists of a value network for approximating the state-value function $v(s)$ and a policy network for approximating the state-action function $q(a)$. These two networks are both fully connected,

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 θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 2.2: Pseudo-code of the PPO-algorithm [40]

deep neural networks. The policy network is responsible for suggesting an action based on the current state the environment is in. The value network estimates the expected return at the end of the trajectory based on the current state [6]. The general principle of the PPO-algorithm is shown in fig. 2.2. Additionally, an action-advantage function estimates the advantage between taking an action in the current state over following the current policy. Herefore, PPO utilizes the generalized advantage estimator (GAE) [39]. This increases the likelihood of taking favorable actions yielding high rewards over unfavorable actions [40]. In order to avoid too large policy updates, which may lead to instabilities, the gradient for updating the policy is clipped as shown in figure 2.3.

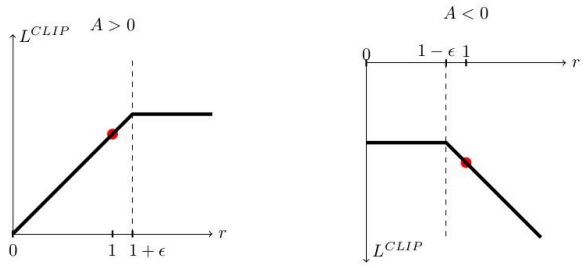


Figure 2.3: Clipping of advantages [40]

2.2.2 Disadvantages of model-free DRL algorithms

Model-free DRL algorithms are generally known for running very stable while yielding good results at the same time. However, a main disadvantage is their poor sample efficiency causing long runtimes and high computational costs for more complex applications, since past experiences of interactions with the environment can not be re-used over the course of the training [18].

Another issue which all DRL algorithms are affected by is the dependency of the performance on the initialization. As discussed in the previous section, DRL-algorithms are non-deterministic, an unfavorable initialization of the model weights and biases may increase the required runtime for reaching a satisfying result or in some cases may lead to no learning progress at all. This behavior is shown in fig. 2.4 for TRPO in a sparse reward environment conducted by [17]. The thick line marks the median reward while the shaded area denotes the corresponding standard deviation. Clearly it can be seen that the reward can cover a wide range from high rewards down to a point of complete failure. To counter this issue, in practice the rewards a generally averaged over multiple different initialization (seed values) in order to account for a too favorable or unfavorable initialization of the models.

While the aforementioned issues affect model-free DRL algorithms in general, when using DRL for real-life applications or physical systems, there arise additional problems. The simulation of physical systems, e.g. in CFD, requires significantly longer runtimes and computational resources as for example generating images of a video game. This currently leads to a strong

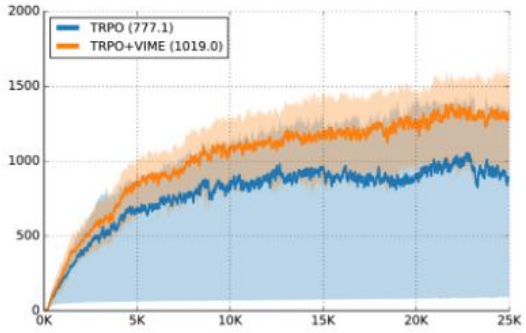


Figure 2.4: Performance of TRPO and VIME on the *Walker2D* locomotion task [17]

limitation of the available applications of DRL for flow control applications and even for those rather simple cases the CFD part makes up 99.7% of the total runtime [32].

Despite the high computational demands and costs involved, the exploration of the agent during training is another issue. For technical applications such as autonomous cars, random actions are potentially harmful and therefore a major safety concern. However, in order to achieve high rewards and a robust policy, exploration is indispensable. These disadvantages of MF-DRL algorithms result in the necessity of modeling the real environment and then let the agent interact with these environment models. This approach is called model-based DRL and will be discussed in the next section.

2.3 Model-based DRL

Model-based DRL (MB-DRL) aims to train additional models for learning the system dynamics of the real environment, which then can be used as surrogate models for training the agent [50]. In contrast to model-free algorithms, the sample efficiency is significantly better since experiences of the interactions with the real environment can be re-used for training the environment models. However, the applicability of those algorithm is highly dependent on the model accuracy and complexity of the environment making it a challenging task [1].

2.3.1 Challenges of model-based DRL

Training sufficiently accurate environment models is still a major challenge in model-based DRL as a result of aleatoric and epistemic uncertainties. Aleatoric uncertainties are referred to as stochastic model-inherent uncertainties caused by the non-deterministic behavior of deep neural networks [4]. Epistemic uncertainties on the other hand are systematic uncertainties resulting from a limited amount of training data [50]. As a consequence, the environment models can not see all possible states during training leading to inaccurate predictions of the environment state for a given unseen input state. These model inaccuracies are further exploited by the policy resulting a good performance in the simulated environment but a worse performance when tested in a real environment, referred to as model-bias [50]. This exploitation leads to a policy optimization to regions where no or only sparse training data is available causing unfavorable performance and stability issues [21]. With increasing length of the horizon these uncertainties are amplified as a consequence of error propagation, since the predicted state of the environment model s_t is used as input into the environment model in order to predict s_{t+1}.

Although MB-DRL suffers from the same initialization problems their MF counterpart as already shown in fig. 2.4, this issue deteriorates for MB-DRL due to the additional environment models which are initialization-depended as well, leading to a further destabilization of the training process [7]. The stability of the training is from paramount importance, especially when combining

DRL and CFD due to the stability of the flow simulation itself. In contrast to e.g. video games, the performance of the agent highly influences the availability of training data. Due model-bias an unfavorable policy may result in actions which cause the flow simulation to diverge. In such cases, there exists no guarantee of generating enough or any training data at all to update the environment models. A major challenge of modeling physical systems is therefore the correct handling of such exceptions and dealing with incomplete or no training data.

In order to solve or at least mitigate the issue of model-bias and model inaccuracies there have been developed a variety of different techniques. One commonly used approach is to utilize a model ensemble (ME) instead of only one single environment model, e.g. in ME-TRPO [21]. The ensemble is here trained on the same training data, but each model in the ensemble is initialized differently and the order in which the training data is fed into the environment models differs. For predicting the next state s_{i+1}, a model is randomly chosen out of the ensemble. This technique has proven to be able to reduce the aleatoric uncertainties and the model-bias significantly while only little adjustments to existing DRL algorithms are necessary. An example of the principle of a training using model ensembles instead of a single model is depicted in algorithm 2.5. From this figure it can be seen that the only real difference is the initialization of an ensemble instead of a single environment model while the rest of the implementation remains overall the same. Due to this fact, the present thesis utilizes an approach similar to the one shown algorithm 2.5.

Another possibility is modeling of the uncertainties itself. This can e.g. be achieved by using Bayesian networks as it is done in [43] and [19] and in the PETS algorithm [4]. However, the complexity of such an implementation is significantly higher than the aforementioned method. Another, relatively new approach is to train additional models on the error between the real and the model-generated trajectories and then correct the model-generated trajectories by this error. This so-called domain adaption is utilized in AMPO presented in [42, 48] and [12]. A largely simplified version of this idea was implemented and tested in this thesis as well and will be discussed in section 5.2.5.

Algorithm 2 Model Ensemble Trust Region Policy Optimization (ME-TRPO)

Initialize a policy π_θ and all models $\hat{f}_{\phi_1}, \hat{f}_{\phi_2}, ..., \hat{f}_{\phi_K}$.
Initialize an empty dataset $\mathcal{D}$.
repeat
  Collect samples from the real system f using π_θ and add them to D.
  Train all models using $\mathcal{D}$.
  repeat ▷ Optimize π_θ using all models.
    Collect fictitious samples from $\{\hat{f}_{\phi_i}\}_{i=1}^{K}$ using π_θ.
    Update the policy using TRPO on the fictitious samples.
    Estimate the performances $\hat{\eta}(\theta; \phi_i)$ for $i = 1, ..., K$.
  until the performances stop improving.
until the policy performs well in real environment f.

Figure 2.5: Pseudo-code of the ME-TRPO algorithm [21]

Chapter 3

Numerical method

The following section presents the numerical setup used in this thesis. Since this thesis follows up on the work done in [6, 9] and [41]; the flow control problem in general remains the same as in the previous work. However, there are some minor adjustments withing the numerical setup as well as the discretization of the numerical domain which will be presented in the following. A comprehensive overview of the numerical setup can be found in [6].

3.1 Flow past a cylinder

The incompressible, two-dimensional flow past a cylinder is a frequently used test case for flow control problems and benchmarks, e.g. in [36], since the theoretical foundations of the present flow characteristics are well established. An additional advantage of this test case is the comparably low computational cost allowing to conduct a vast variety of simulations while keeping the required runtime to a minimum level.

In the original work conducted in [32, 8] and [10], the flow control is achieved by an injection of mass flow using plasma actuators. These actuators are controlled by deep neural networks which where trained using MF-DRL. In contrast to this form of flow control method, instead of mass flow injection the cylinder is rotated with an angular velocity ω depending on the current state, similar to the work presented in [47]. The angular velocity ω is sampled from a beta-distribution outputted by the policy network and bound by the interval $\omega \in [-5, 5]\, rad/s$. Further, the reward function is defined as

$$r_t(s_t, a_t) = 3 - (c_D + 0.1|c_L|) \tag{3.1}$$

hence the goal is to minimize the forces acting on the cylinder. For more detailed information the reader is referred to [6] and [9].

3.1.1 Numerical setup of the flow problem

The geometry of the numerical domain remains the same to all previous work in order to ensure comparability. In contrast to the aforementioned thesis projects, additional to the forces at the cylinder surface, pressure values are logged throughout the simulation at the marked positions in figure 3.1. These additional information on the current state are used as input into the policy network. Providing such information on the flow may reduce the issue of over-fitting as a consequence of low-dimensional data as occurred in [41] when moving towards a MB-training. There are currently 12 probes placed in the wake of the cylinder, in the following denoted as states.

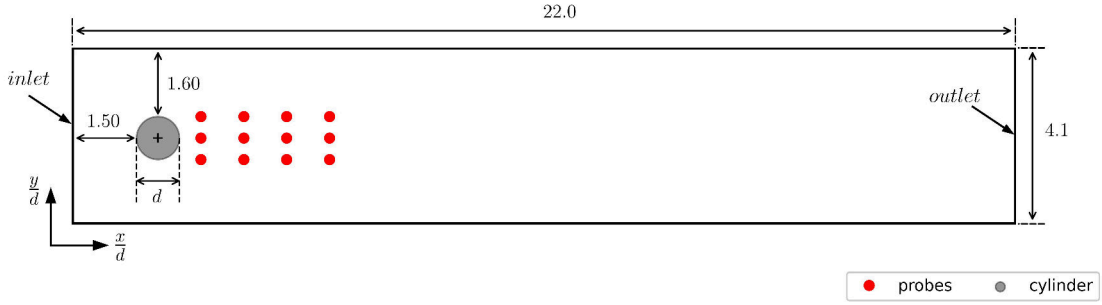


Figure 3.1: Geometry of the numerical domain, *probes* denotes the sensors placed in the wake of the cylinder

Incompressible Navier-Stokes equations

The two-dimensional flow past a cylinder can be described with the Navier-Stokes equations for incompressible flow. The Navier-Stokes equations are obtained in their non-dimensional form and can be found in A.1 along with the corresponding quantities for non-dimensionalization.

Initial- and boundary conditions

The boundary conditions remain the same as in all previous work, but for the sake of completeness they shall be presented here briefly. At the upper- and lower boundaries of the domain, a no-slip condition is applied as $\mathbf{u}^*(x^*, y^* = 0) = \mathbf{0}$ and $\mathbf{u}^*(x^*, y^* = h/d) = \mathbf{0}$. The same condition is enforced to the cylinder surface as well. A parabolic inflow velocity profile is defined as

$$\mathbf{u}^*(x^* = 0, y^*) = \frac{4U_m y(h-y)}{h^2} \tag{3.2}$$

at the inlet, where $U_m = 1.5\,\mathbf{u}_\infty$ at $y = h/2$ with h denoting the height of the domain. The velocity in $y-$direction is set to $\mathbf{v}^* = 0$. At the outlet, zero-pressure as well as a zero velocity gradient normal to the outlet is applied.

The cylinder diameter $d = 0.1m$ is used to non-dimensionalize all spacial quantities and remains constant. Throughout this thesis the Reynolds number is always kept constant at $Re = 100$, except for section 5.4. With a kinematic viscosity of $\nu = 1 * 10^{-3}$, this leads to a free stream velocity at the inlet of $\mathbf{u}_\infty = 1m/s$.

The initialization of the flow field can be divided into two parts. In a first part, the transient phase up to a quasi-steady flow field is computed. The flow field is initialized with zero at $t^* = 0$ and the flow remains uncontrolled until reaching a quasi-steady state at $t^* = 40$. This simulation is referred to as the *base* case. As described in [32] and [9], multiple simulations are run simultaneously within every episode in order to accelerate the training process. With the flow control starting in the quasi-steady state, all simulations are initialized with the *base* case at $t^* = 40$ and then run up to a specified end time. This initialization avoids the necessity to compute the transient phase for each simulation run in the training saving a significant amount of computational resources.

3.1.2 Domain discretization

The numerical domain is discretized using the built-in discretization tools *blockMesh* of the open-source CFD software *OpenFOAM*. The *blockMesh* utility discretizes the domain into hexahedral blocks as depicted in fig. 3.2. A grading in $x-$direction is used to coarsen the mesh downstream in order to reduce computational requirements.

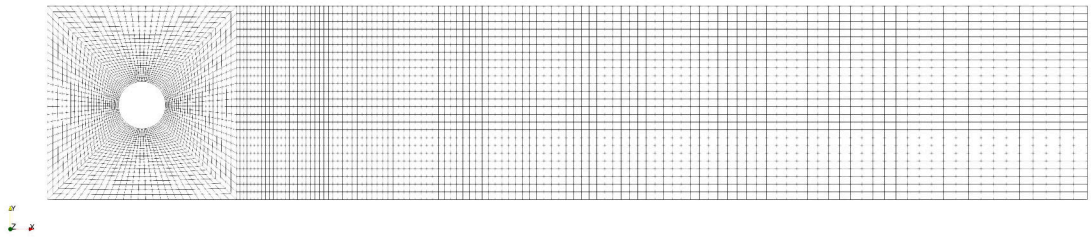


Figure 3.2: Spatial discretization of the flow problem

The domain is decomposed into two subdomains during runtime, which are solved simultaneously in order to accelerate the simulation. Hereby, the subdomains contain roughly an equal amount of nodes. The OpenMPI library (compare section 3.2.6) ensures the exchange of information of the flow field at the decomposition interface. The flow field itself is solved with the incompressible, transient solver *pimpleFoam* [25]. An example tutorial of solving the flow past a cylinder is provided in the OpenFOAM documentation [24].

Mesh dependency studies have already been carried out by [6] and [9]. However, in order to save computational resources, the mesh used in this thesis was coarsened in contrast to these prior studies. A further coarsening of the mesh reduces the required runtime per simulation significantly while the deterioration of the accuracy can be seen as neglectable. In order to quantify the mesh requirements with respect to the minimal number of grid points, the correlation within the pressure values at the probe locations, and to the coefficients c_L and c_D were evaluated for different mesh refinement levels. The correlation coefficient R between two variables is hereby defined as

$$R_{xy} = \frac{cov(x, y)}{\sigma_x \sigma_y} \tag{3.3}$$

with $cov(x, y)$ denoting the covariance and σ the standard deviation. The correlation coefficient is bound by the interval $R \in [-1, 1]$ and can be used to calculate the linear correlation between two variables.

Figure 3.3 shows the differences within these correlation maps ΔR_i for the different refinement levels of the mesh. Here, the number of grid points is increasing from the lowest refinement level 0 to the highest level 3. It can be seen clearly that the correlations for refinement levels 0 to 1 change significantly, indicating a strong dependency of the mesh on the flow solution. For the refinement levels 1 to 2, there is only a minor difference within the correlations, while for level 2 to 3 the differences are neglectable. Since the goal was an optimization of the required computational resources, for all further computations the refinement level 2 is seen as sufficiently mesh independent.

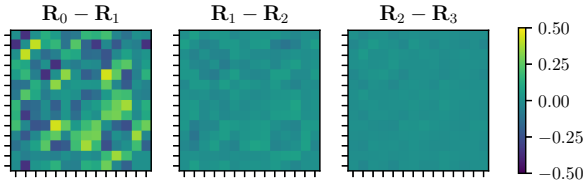


Figure 3.3: Differences within the correlation heat maps for different grid refinement levels[1]

[1]These studies where conducted by my supervisor Dr.-Ing. Andre Weiner and by the time this report was written not published yet. The usage of this data was permitted by Mr. Weiner.

N_{points}	max. AR	max. skewness	Δt	mean CFL	max. CFL
11076	2.33	0.741	$5 * 10^{-4}$	0.0459	0.32

Table 3.1: Characteristics of the mesh and simulation used in this thesis (CFL numbers are for the uncontrolled case)

The final number of mesh points as well as additional information regarding the mesh quality and CFL number can be obtained from table 3.1.

3.1.3 Adjustments for higher Reynolds numbers

Towards the end of this thesis the Reynolds number was increased to $Re = 500$ and $Re = 1000$ in order to test the abilities of the environment models to adapt to new system dynamics. Therefore the mean inlet velocity was increased to $u_\infty = 5m/s$ and $u_\infty = 10m/s$, respectively. As a consequence of the higher Reynolds number, the mesh requirements with respect to the number of mesh cells increase as well, consequently a new mesh dependency study was conducted. A refinement of the mesh and increasing the inlet velocity leads to the necessity of decreasing the numerical time step in order to fulfill CFL-condition, which is defined for a two-dimensional flow as

$$CFL = u\frac{\Delta t}{\Delta x} + v\frac{\Delta t}{\Delta y} \leq CFL_{max} \tag{3.4}$$

the maximal CFL number depends hereby on the temporal discretization scheme used. For $Re = 500$, the time step was decreased to $\Delta t = 1 * 10^{-5}s$ in order to fulfill the CFL condition while ensuring that the amount of interactions between agent and environment remains constant for the same dimensionless time interval.

In order to conduct the mesh dependency study, the numerical setup presented in [49] was used as orientation while the grid for $Re = 100$ was used as a starting point. The results of this study are presented in fig. 3.4, for both c_L and c_D an overall convergence behavior can be seen when refining the mesh, although the c_L-value for $N_{points} = 43350$ is decreased compared to $N_{points} = 21456$. Increasing the number of cells from 11076 to 21456 results in an increase of c_L by 77.89%, a further increase to $N_{points} = 43350$, however, leads to a decrease of c_L from $c_L = -0.0210$ to $c_L = -0.0427$. With $N_{points} = 84490$, c_L increases again to a value of $c_L = -0.0215$ indicating a convergence. The course of c_D yields a convergence behavior as well, starting with $c_D = 3.492$ for $N_{points} = 11076$. An increase of the mesh cells to $N_{points} = 21456$ causes a decrease of 4.2%. An increase of N_{points} to 43350 corresponds to a decrease of c_D by 2.9% while a further refinement to $N_{points} = 84490$ only shows a minor decrease by 1.3%. Since c_D contributes the major part of the rewards, an accurate computation of c_D is seen as decisive factor for determining the number of required mesh cells, on the other hand the runtime

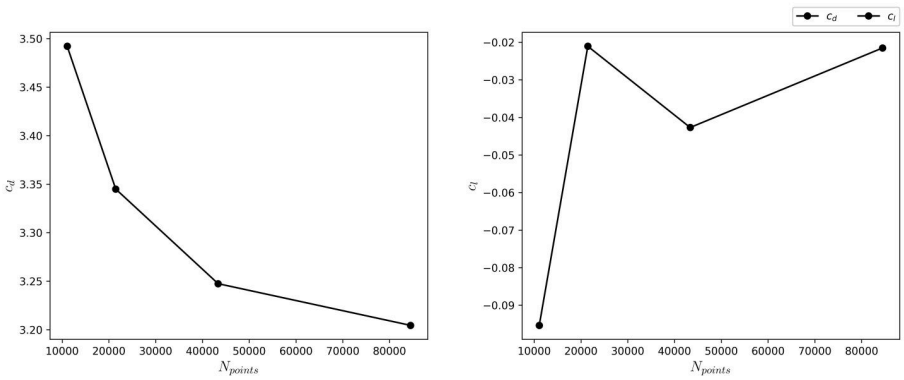


Figure 3.4: Grid convergence study for $Re = 500$

increases significantly with increasing number of mesh cells; therefore $N_{points} = 43350$ is chosen. The temporal evolution of the corresponding lift and drag coefficients with respect to the number of mesh cells can be found in the appendix, section A.2.

The chosen setup leads to an average Courant number of $CFL_{avg} = 0.103$ and a maximum Courant number of $CFL_{max} = 0.757$. The mesh dependency study conducted for $Re = 1000$ can be found in the appendix, section A.2; the chosen number of mesh cells remains the same as for $Re = 500$. The time step was decreased further to $\Delta t = 5 * 10^{-5} s$ to ensure the stability of the simulation while maintaining a constant sample frequency as already discussed. The average and maximum CFL numbers are similar to the ones for presented for $Re = 500$.

3.2 Libraries and computational resources

The following section gives a brief overview of important libraries required for running a training with the PPO-algorithm. This overview is by no means comprehensive, but for ensuring the ability of reproducing the results presented throughout this thesis, the most important software and their versions are presented.

3.2.1 Phoenix HPC

Most of the trainings are run on an HPC cluster of the TU Braunschweig[2] due to the high computational costs of running a vast variety of parameter studies. For comparison with respect to runtimes as well as for testing purposes, additional trainings are run on a local machine equipped with an *Intel® Core™ i7-11800H* CPU with 8 cores and 32 GB of RAM. It is important to note that all trainings, for the PPO-algorithm as well as for training the environment models, CPUs are used instead of GPUs.

3.2.2 AWS

Amazon Web Services Inc. (AWS) is a cloud computing provider hosted by Amazon Inc.[3] When conducting a training on the *Phoenix* cluster, instabilities occurred as will be described in section 5.2.5. To investigate a possible dependency of the stability of the training routine on the available computational resources, towards the end of this thesis the training routine was migrated to *AWS*. The overall training procedure and required libraries, however, remain the same.

3.2.3 PyTorch

PyTorch is an open-source framework for deep learning in Python developed and maintained by the *PyTorch foundation* [29]. *PyTorch* provides an API for Python as well as C++, while the backend is implemented in C++ making it a fast, but at the same time user friendly library. Additional to tools for deep learning, *PyTorch* implements a variety of methods for data processing and optimization frameworks. The optimization frameworks used in this thesis are the *Adam* optimizer [20] and the *AdamW* optimizer [22], which is an improved version of the *Adam* optimizer. In this thesis, *PyTorch v1.12.1* is used, more information on the underlying principles of *PyTorch* can be found in [27].

3.2.4 OpenFOAM

OpenFOAM is an open-source CFD software, which is able to solve a wide range of fluid mechanical problems [24]. In this thesis, *OpenFOAM v2206* is used for discretization as well as

[2] Phoenix cluster of TU Braunschweig: `https://www.tu-braunschweig.de/it/dienste/21/phoenix`, last access: 09.11.2022

[3] `https://aws.amazon.com/`, last access 18.12.2022

solving the flow field. *OpenFOAM* is written in C++ making it possible for coupling with other frameworks, e.g. *PyTorch*, or implementing custom functionalities and extensions. Additionally to the CFD software itself, *OpenFOAM* provides a variety of tutorials and test cases, one of which is the incompressible flow past a cylinder similar to the flow problem presented in section 3.1.1 [24].

3.2.5 Apptainer

Apptainer (former *Singularity*) is an open-source framework for containerization of applications and software[4]. In contrast to other frameworks, e.g. *Docker*, *Apptainer* was designed for HPC applications and can therefore for example be executed without root privileges.

3.2.6 OpenMPI

OpenMPI is an open-source library for inter-process communication (message passing interface) and generally required for high-performance computing [46]. In this thesis, *OpenMPI v4.1* is used. More information on *OpenMPI* can be found in [11].

3.2.7 Drlfoam

This thesis utilizes *drlfoam*[5], a framework for applying DRL for flow control problems in CFD. *Drlfoam* currently provides the MF-training routine for controlling the flow past a cylinder, developed based on [6] and [9], in a standardized interface. It incorporates the aforementioned libraries in order to make DRL in CFD applicable as well as possibilities to extend the current implementation by additional environments or methods.

[4] `https://apptainer.org/docs/user/main/`, last access: 13.12.2022

[5] drlfoam is publicly available under: `https://github.com/OFDataCommittee/drlfoam`

Chapter 4

Modeling the CFD environment

Due to the variety of different possible approaches for modeling the CFD environment with deep neural networks, prior training an agent with environment models the accuracy and practical implementation of those models are to be tested. In this section, first the current implementation of the MF-training is benchmarked in order to derive requirements for stability and performance of a MB-training. In a next step, the data of the MF-training is used for evaluating the general feasibility of approximating the CFD environment with models. The work done by [41] is therefore used as a starting point. It is important to note that all trainings in this section are performed at $Re = 100$, the hyperparameter settings of the policy and value network of the PPO-algorithm remain constant throughout this thesis and can be found in the appendix, section A.1. This section concludes with a possible method for approximating the CFD environment which is then used as a starting point for the integration into *drlfoam*.

4.1 Influence of the buffer size and trajectory length for MF-training

In a first step, the goal was to investigate the influence of the buffer size and trajectory length on the received rewards and general performance of *drlfoam*. These information are then utilized in order to derive requirements for a model-based training routine with respect to stability of the training routine.

The model-free training routine as it is currently implemented in *drlfoam* has mainly the two parameters buffer size and trajectory length, which influence the performance of the agent in terms of achieved rewards. The buffer size b denotes the number of simulations (environments) run parallel in each episode and therefore determines the number of trajectories available in each episode. The trajectory length l denotes the horizon, considering the sample frequency of $100Hz$, this means $1s$ of physical simulation time corresponds to 100 interactions between agent and environment for each trajectory at each episode within the buffer. Considering further a vortex shedding frequency of $f(c_L) = 2.67Hz$ (see section 4.2.4), this results in ≈ 37.45 interactions per period of c_L. Consequently, each second of trajectory length corresponds to 2.67 periods of vortex shedding for c_L if a sample frequency of $100Hz$ is applied.

Both the buffer size and trajectory length are global variables set at the beginning of the training and remain constant over the course of the training. The number of processes run in parallel is always the same as the buffer size in order to ensure that all processes finish at roughly the same time. Further, all cases are run for 80 episodes since the previous work done by [6] and [9] indicated a convergence of the rewards within the first 80 episodes. Increasing the number of episodes to > 80 did not yield any significant improvements as will be discussed at the end of

Parameter		**min.** μ	**max.** μ	**min.** σ	**max.** σ
runtimes	[s]	5071	41747	66.4	8845.7
rewards	[-]	-0.20	-0.05	0.05	0.08
c_L	[-]	3.028	3.152	0.030	0.063
c_D	[-]	0.003	0.156	0.306	0.585

Table 4.1: Values used for the min- max scaling of c_L, c_D, runtimes and the rewards

this chapter. To counteract the dependency of the performance on the initialization as discussed in section 2.2.2, all trainings are averaged over three different seed values.

Figure 4.1 shows the resulting rewards for each combination of buffer size and trajectory length. As presented in section 3.1, the reward consists of c_D and c_L, for which the heatmaps can be found in the appendix, section A.3. For a better comparison, the rewards are scaled to an interval of $[0, 1]$ using a min- max scaling, consequently $r^* = 0$ denotes the lowest and $r^* = 1$ the highest rewards averagely received throughout the training. Further, the reward was averaged over 3 seeds and all episodes, where μ refers to the mean value and σ to the corresponding standard deviation. Since the absolute values for σ are generally several orders smaller than the mean values, μ and σ are scaled independently of each other. This procedure applies to fig. 4.2 and 4.3 as well. The absolute min- and max-values used for the scaling can be found in table 4.1.

From fig. 4.1, mainly two trends can be observed. Increasing the trajectory length leads to a significant increase in the received rewards from $r^* = 0.0...0.2$ for $l = 1s$ up to $r^* = 0.6...1.0$ for $l = 8s$. The increase of the trajectory length from $l = 1s$ to $l = 2s$ yields with up to 50% for $b = 10$ the largest improvement. This behavior is caused by the fact, that is takes approximately one second for the agent to reduce c_L and c_D starting from the uncontrolled flow with $c_L \in [-1, 1]$ and $c_D \approx 3.19$ to values yielding high rewards. If the trajectory ends after $l = 1s$, however, then the agent is not able to explore regions with low c_D and $|c_L|$ values. With increasing trajectory length, the relative amount of this initial phase decreases leading to increasing rewards.

Secondly, with respect to the rewards, the buffer size has only little influence on the overall results. For small buffer sizes such as $b = 2$ or $b = 4$ the standard deviation tend to be higher than for larger buffer indicating stability issues for small buffer sizes. As shown in fig. 4.4b, for $b = 2$ with increasing trajectory length the course of the achieved rewards tends to become unstable, while this behavior could not have been observed for $b = 10$ as presented in fig. 4.4a.

The available computational resources are another important factor, represented by the required

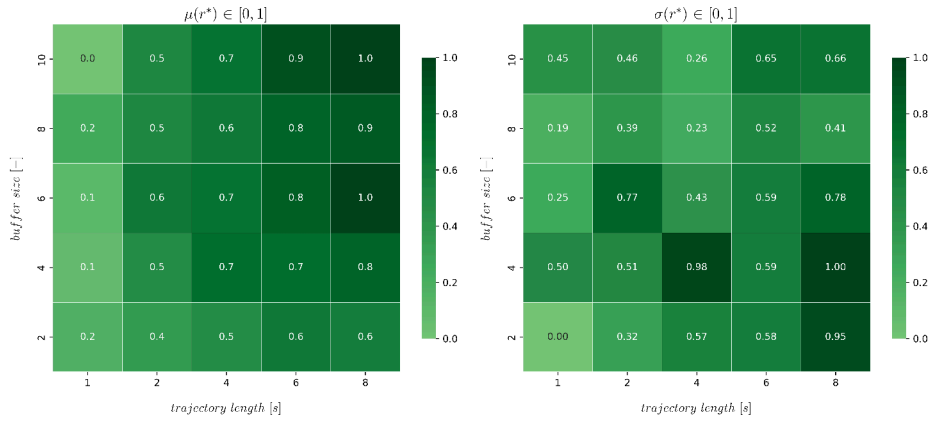


Figure 4.1: Average rewards received with respect to the buffer size and trajectory length

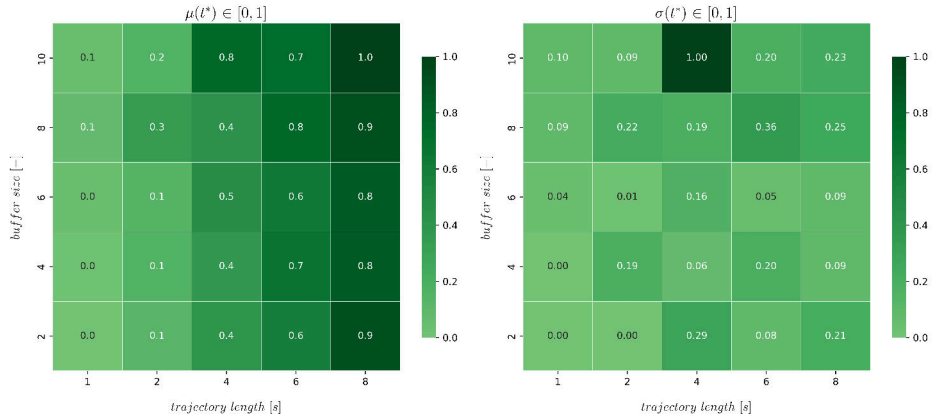


Figure 4.2: Average runtimes with respect to the buffer size and trajectory length

runtime for each training. As shown in figure 4.2 the runtimes largely depend on the trajectory length while the influence of the buffer size is neglectable. This is caused by the fact that the number of cores was always chosen to be equal to the buffer size and therefore all required simulations for each episode are run parallel. Provided that enough computational resources are available, the trajectory length remains consequently the only parameter determining the runtime. From fig. 4.2, it can be seen that for short trajectories, e.g. $l = 1s$ or $l = 2s$ the increase is neglectable, however, for longer trajectories, the runtime increases significantly. It is important to note that the standard deviation for trainings run with $b = 10$ and $l = 4s$ is remarkably higher than for all other cases. This is probably caused by a high traffic on the HPC at the time these trainings where run. The standard deviation of the runtimes are generally increasing with increasing buffer size, since each simulation within the buffer allocates 2 cores. At some point the resources available are not sufficient for running all simulations parallel without the necessity of holding processes for their execution. It is therefore concluded that a further increase of the buffer size to $b > 10$ is not beneficial considering the low impact of the buffer size on the rewards.

Figure 4.1 and 4.2 were used in a last step to find possible optimal combinations of buffer size and trajectory length. Since it is the overall goal to minimize the required runtimes while at the same time maximizing the rewards, the optimization problem can be formulated as minimization of the superposition of runtime and inverse of rewards, which is depicted in fig. 4.3. It can be seen that clearly there is no real optimal combination of b and l since both the rewards and the runtime increase with increasing trajectory length. The buffer size on the other hand has only

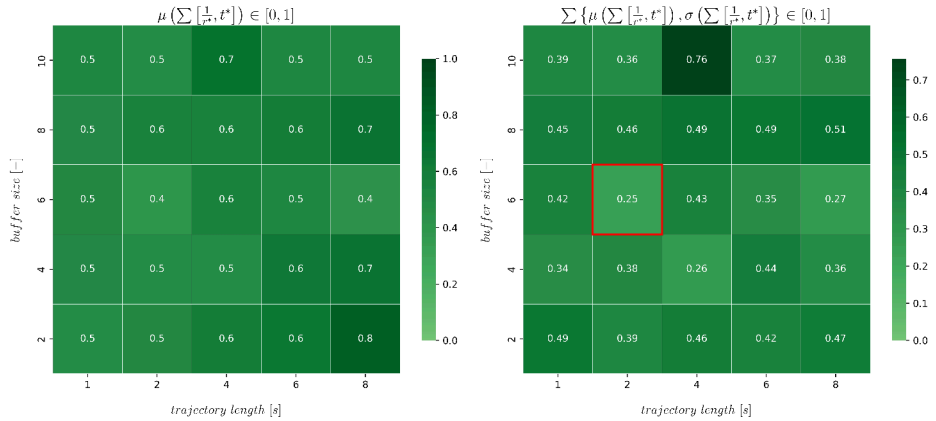
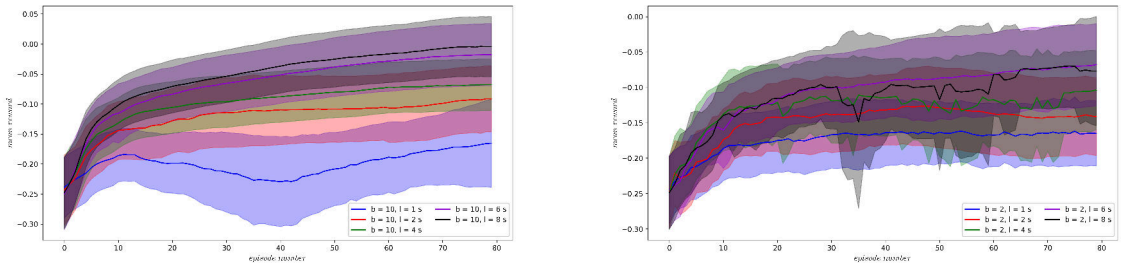


Figure 4.3: Optimum of the runtime and received rewards, the red rectangle marks the minimum $min\left\{\sum\left(\mu\left[\frac{1}{r^*}, t^*\right], \sigma\left[\frac{1}{r^*}, t^*\right]\right)\right\}$



(a) buffer size $b = 10$ **(b)** buffer size $b = 2$

Figure 4.4: Rewards received throughout the training for buffer sizes $b = 2$ and $b = 10$ with respect to the trajectory length

a neglectable effect on both the runtime and rewards, provided there are enough computational resources available. Nevertheless, fig. 4.3 shows that small buffer sizes of $b = 2$ or $b = 4$ in combination with long trajectories, e.g. $l = 6s$ or $l = 8s$ are rather unfavorable.

Despite runtime and rewards, the stability of the training itself is a decisive factor for determining requirements for a MB-training. In this context, stability means not only the execution of the training without any exceptions or errors but a monotonic, continuous increase of the rewards over the course of the training. Figure 4.4a and 4.4b depict the received rewards with respect to the episode number for the buffer sizes $b = 2$ and $b = 10$ using different trajectory lengths. The plots for buffer sizes $b = 4, 6$ and 8 can be found in the appendix, section A.3. For a buffer size of $b = 10$, except for $l = 1s$, the rewards increase monotonically over the course of the training. The rewards for $l = 1s$ only shown a minor improvement over the course of the training due to the transient phase at the beginning of the trajectory as already described. Further, as expected with increasing trajectory length the rewards increase as well. However, for a buffer size of $b = 2$ the received rewards are generally worse than for $b = 10$. For $b = 2$ in combination with $l = 4s$ and $l = 8s$, the training yields discontinuous rewards suggesting stability issues for long trajectories in combinations with small buffer sizes.

Another trend observed can be seen in fig. 4.5a and 4.5b, which depicts again the total rewards received throughout the training. For $b = 10$ the standard deviation within the received rewards is generally smaller than for $b = 2$. Further the convergence behavior for trajectory length of $l \geq 6s$ can be observed. For $b = 2$ and $l = 8s$ the rewards are starting to decrease again while the standard deviation is significantly higher than for all other cases confirming possible stability issues when combining long trajectories with small buffer sizes.

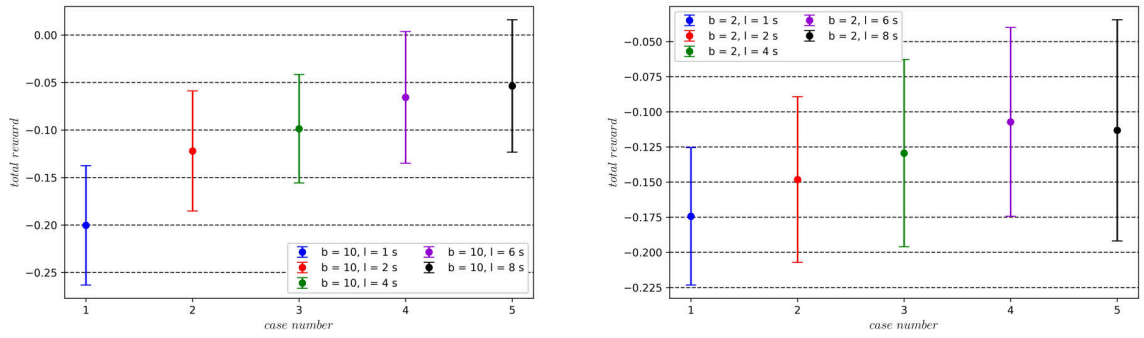
These results can now be used to derive requirements for a model-based training routine. In summary it was found that small buffer sizes, e.g. $b = 2$ or 4 yield only neglectable worse results than e.g. $b = 10$, but may cause stability issues during training. Considering that environment models require a reasonable amount of data for training and validation as well as additional data to sample initial states for the model-generated trajectories from, a larger buffer may be beneficial. The buffer size has shown no significant impact on the required runtime making an increase in buffer size an efficient way of generating larger amounts of training data for the environment model. To account for possible trainings on a local machine a MB-training should therefore be able to run stable with a buffer size of $b = 8$ as well. Regarding the trajectory length, due to a convergence behavior and with respect to the required runtime, $l = 6s$ seems to be sufficiently long. On the other hand the trajectories should have at least a length of $l > 1s$ in order to achieve a satisfactory performance in terms of rewards. For running a training on a local machine, $l = 2s$ is in general a good compromise in order to conduct a full MF-training in a reasonable amount of time. For an HPC application $l = 6s$ yields the highest rewards, however, even advanced MB-DRL algorithms, such as MBPO, are only able to confidentially predict up

to 200...500 subsequent time steps sufficiently accurate [18]. A maximum trajectory length of $l = 4s$ is therefore seen as a realistic goal for a MB-DRL algorithm.

Conclusively, the convergence behavior of the PPO-training was investigated. A training with $b = 10, l = 8s$ was run for 200 episodes in order to make sure that the observed convergence behavior was not just a local saddle point but marked the convergence boundary. It was found that the training converged within the first 80 episodes as it could have been observed in all the prior trainings. The rewards, however, started to oscillate at episode ≈ 175, two out of five trainings even were aborted due to a divergence of the CFD simulation. This indicates that there exist an upper limit with respect to the number of episodes before instabilities are starting to arise. The corresponding rewards with respect to the episode number can be found in the appendix, fig. A.7.

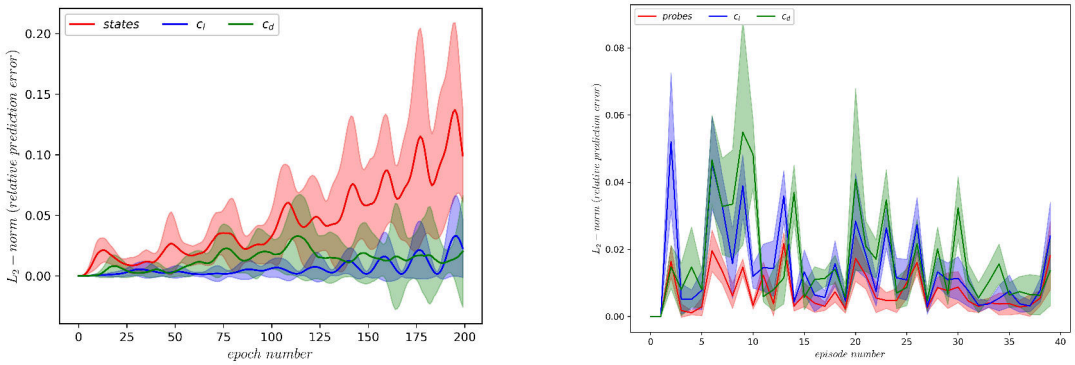
4.2 Overview of possible approaches for MB-DRL

The work of [41] was taken as a starting point for developing environment models for approximating the flow simulation. In contrast to [41], the present thesis utilizes fully connected neural networks instead of recurrent ones, also additional to c_L and c_D, the pressure values at the defined probe locations in the wake of the cylinder are used as additional information on the current state. Modeling the trajectories of the probes is an additional requirement since these values function as input for the policy network. In a first step, one environment model is trained on the available training data despite taking the episode number in which the data was created into account. Based on the obtained results, the prediction accuracy is then gradually improved using more complex models and techniques. Since the overall prediction accuracy of these environment models is to be evaluated and improved, the data for training, validating and testing the environment models is taken from a MF-training with the buffer size of $b = 8$ and a trajectory length of $l = 2s$. Out of these training data, further only the first 40 episodes are taken, because within the first 40 episodes the trajectories have the most variance, which means that for episodes $e \geq 40$, the rewards underlay only minor improvements and the trajectories in general are distributed homogeneously. For assessing the accuracy of the environment models, it is sufficient to evaluate their performance within the first 40 episodes. If the models are able to perform accurate predictions over a wide range of differently shaped trajectories (data with high variance), it can be concluded that they also perform well at data with low variance, namely for episodes $e > 40$. This concept ensures at the same time that the computational costs can be reduced to a minimum, since a vast variety of parameter studies have to be conducted on a local machine.



(a) buffer size $b = 10$ **(b)** buffer size $b = 2$

Figure 4.5: Total rewards received over the course of the training for buffer sizes $b = 2$ and $b = 10$ with respect to the trajectory length



(a) L_2-norm with respect to the epoch number using one global model

(b) L_2-norm with respect to the episode number using one new model each episode

Figure 4.6: L_2-norm of the prediction error for the two different approaches, averaged over all episodes and trajectories within the buffer

4.2.1 One global environment model vs. one new model each episode

In a first step one environment model was trained on the complete data set. The data set was therefore split into training (65%), validation (30%) and test data (5%), where the test data refers to the trajectories which are used to test the trained environment model on. The hyperparameters of the environment model can be found in the appendix, section A.5.2. The model was trained using two subsequent time steps as input, in the following denoted as $N_{t,input}$, while the next time step was the state to be predicted. It is important to note that the model input always consists of the pressure values, ω, c_L and c_D while the model predicts the state, c_L and c_D value of the next time step. During the prediction of the trajectory, the first two time steps of the trajectory were given as input into the model along with the corresponding action taken in the CFD environment. The model then predicted the next state p_i as well as c_L and c_D. This predicted state is then used together with the previous time step and the actions corresponding to those time steps as new model input in order to predict the next time step, until all 200 points of the trajectory are predicted. Since the actions remain the same as for the real trajectories generated in the CFD environment, the model accuracy can directly be assessed by the difference between the real and the predicted trajectories. For a perfect model without prediction errors, the predicted trajectories would be exactly the same as the real ones.

Figure 4.6a depicts the L_2-prediction error, averaged over all trajectories of the test data set with respect to the epoch number. It can be seen that the predictions for c_L and c_D are with up to $\approx$ 3% generally quite accurate, however, the predictions for the probes contain errors up to $\approx$ 14%. The prediction accuracy of the probes is a major concern, because the states are directly used as input into the policy network and therefore responsible for suggesting an action based on the current state. If the predicted state is inaccurate, this is likely to lead to an unfavorable action and increasing model-bias. Further, fig. 4.6a shows the effect of error propagation due to model inaccuracies. With increasing epoch number, which corresponds to the trajectory length, the prediction errors accumulate since the model output of the current time step is used as input for predicting the next time step. This behavior makes the usage of longer trajectories, e.g. $l = 4s$ a challenging task, because as can be seen the error accumulates nearly exponentially.

A direct comparison of a real trajectory generated within the CFD environment with the predicted one is shown in fig. 4.7. It can be seen that the course of c_L can be predicted quite accurately as indicated from the L_2-error. However, as already occurred in the work of [41],

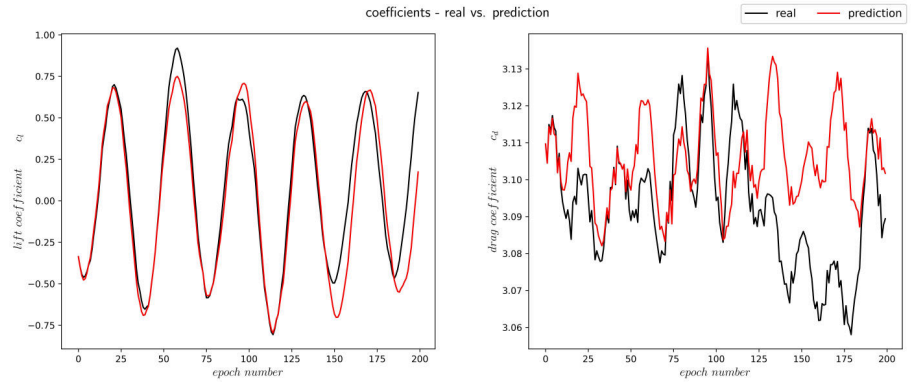


Figure 4.7: Real trajectory generated in the CFD environment in comparison to the predicted trajectory by the environment model using a global model for all episodes (here episode 14 is shown)

the minima and maxima of the c_L trajectory are not predicted accurately enough, further the environment model is not able to remain the same frequency leading to a shift in phase with increasing trajectory length. However, the prediction of c_L overall can be seen as sufficiently accurate when compared to the prediction accuracy of the c_D-trajectory. Although the overall trend of the predicted trajectory matches the real one up to epoch ≈ 100, the prediction accuracy for the trajectory of c_D is significantly worse than for c_L.

Another aspect of using a pre-trained environment model is the flexibility of this approach. Although being rather simple to implement, the model is required to be trained prior running a MB-training. This means at first a MF-training needs to be conducted with the exact same settings as intended for the MB-training, such as Reynolds number, then the environment model needs to be trained and integrated into the MF-training routine. Once this is completed, a MB-training can be run, however, if any parameters of the training change, e.g. trajectory length, number of probes and so on, the complete process of generating training data and training the environment model needs to be repeated. This behavior directly counteracts the upsides of MB-DRL such as sampling efficiency.

A much more convenient and efficient approach is to directly combine the MB- with the MF-training. In this case, the environment models are successively trained every N^{th} episode. These environment models are used to generate trajectories for a certain number of episodes in order to train the agent. As soon as the models become too inaccurate, one episode within the CFD environment is conducted, this data is then used to update the environment models. This approach, in contrast to the prior one, would be independent of any changes within the parameters offering a much greater flexibility.

The feasibility of this approach is tested in a next step. Therefore two subsequent CFD episodes are used in order to train an environment model. The model is then tested on the next CFD episode, e.g. episode 1 and 2 are used for model training and trajectories of episode 3 is used for testing the model. This approach allows to reduce the variance within the data to a minimum, since the trajectories of the test data set are similar to the training data. This approach, however, may lead to an over-fitting of the models since the amount of available training data is reduced significantly compared to the previous approach. Secondly, with respect to an integration into *drlfoam* this may results in a less robust policy, since the variance within the training data is reduced as well.

The resulting prediction error of this new approach can be seen in 4.6b, here the L_2-norm of the prediction error is shown with respect to the episode number, averaged over all trajectories available within each episode. In comparison to the previous approach, the prediction error has

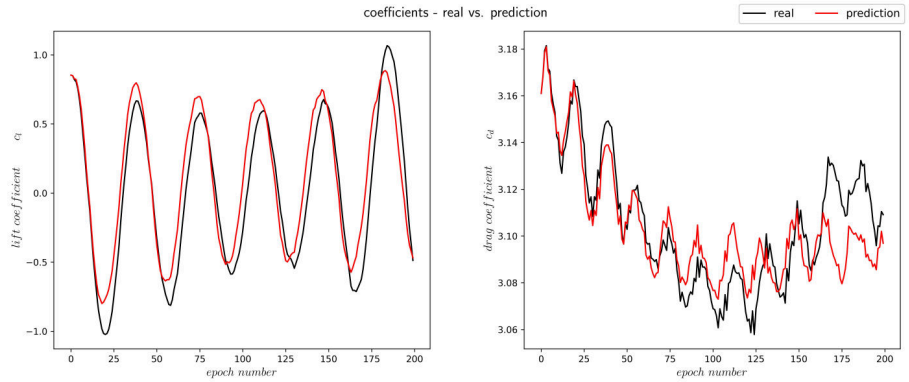


Figure 4.8: Real trajectory generated in the CFD environment in comparison to the predicted trajectory by the environment model using a model trained with trajectories of episode 3 and 4 (here episode 5 is predicted)

improved significantly, if the standard deviation of the prediction error is taken into account. Especially the prediction of the pressure values is now decreased to the same range as the prediction of c_L and c_D. However, at the beginning of the training the prediction accuracy is worse than towards the end of the training. This may be a consequence of the high variance of the training and test data due to random actions taken by the agent at the beginning of the training. These random actions show obviously not clear pattern, making it challenging to learn the underlying system dynamics for an environment model. Although statistically the prediction accuracy improves with increasing episode number, there are still a non-neglectable number of outliers and points with poor prediction accuracy present. These outliers may have a negative impact on the stability of a MB-training, because even if the overall accuracy is acceptable, one or two inaccurate trajectories may steer the policy in an unfavorable region from which it can not recover.

Figure 4.8 shows a predicted trajectory of episode 5 using the episode-wise trained environment model. While the prediction accuracy for c_L remains about on the same level as for the prediction using one global model, it clearly can be seen that the prediction accuracy for c_D improved significantly. Although there are deviations from the real trajectory, up to an epoch of ≈ 150 the prediction error is comparable to the prediction error made for c_L and concentrates mainly on the min- and maxima within the trajectory.

An example of an outlier with a rather poor prediction accuracy on the other hand is shown

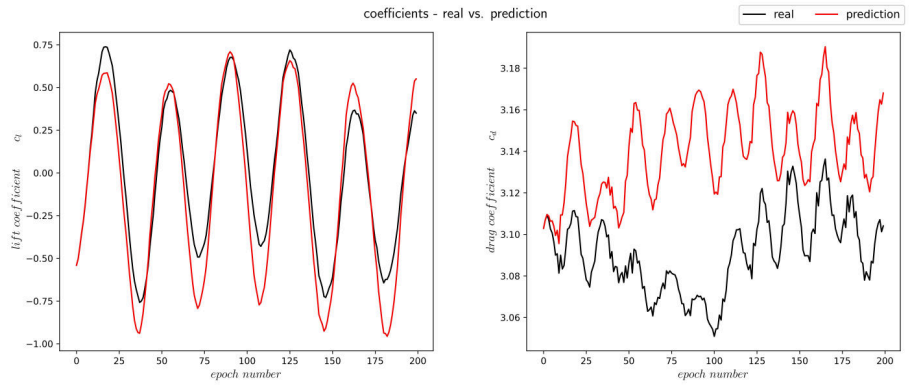
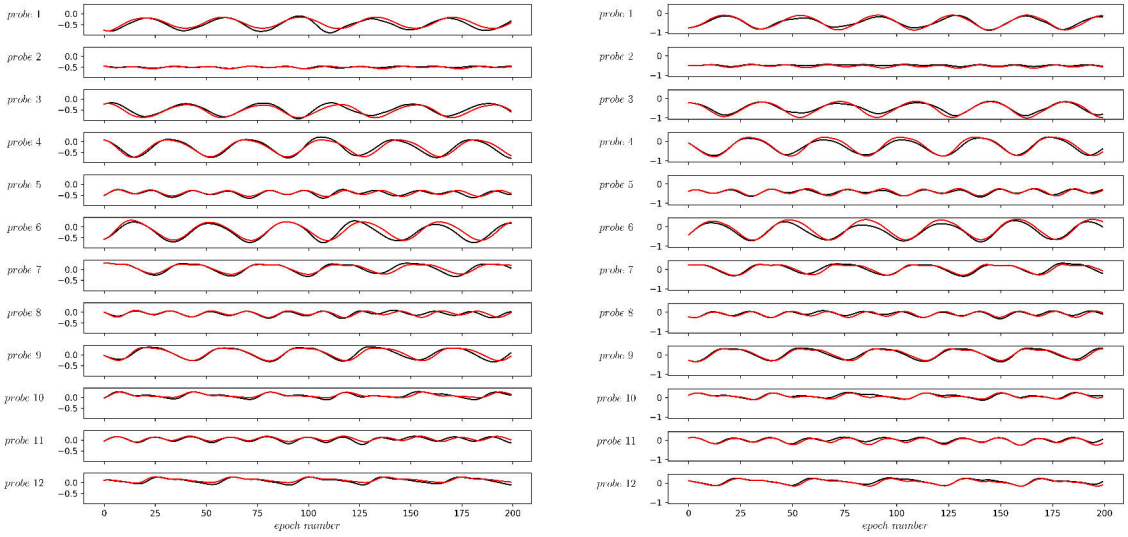


Figure 4.9: Real trajectory generated in the CFD environment in comparison to the predicted trajectory by the environment model using a model trained with trajectories of episode 12 and 13 (here episode 14 is predicted)



(a) real and predicted trajectories using one global model

(b) real and predicted trajectories using a model trained with data from the previous two episodes

Figure 4.10: The trajectories of the probes, corresponding to fig. 4.7 and 4.9

in fig. 4.9. It can be seen that, again, the predictions for c_L are comparable to the accuracy of the previous method, the prediction for c_D, however, is even worse and displays the course of the real trajectory not even qualitatively. But in contrast to the previous approach, for the episode-wise training of an environment model these high errors remain outliers in an overall quite accurate prediction whereas for the previous approach the prediction accuracy for c_D was generally poor.

Lastly, fig. 4.10a and 4.10b depict the prediction of the trajectories of the states. The overall accuracy in both cases is quite well, however, as for the c_L trajectory there exists a phase shifting between the real trajectory and the predicted one. This phase shifting is increasing with increasing trajectory length and more dominant when using one global model as can be seen from fig. 4.10a. The usage of an episode-wise trained model is able to mitigate the phase shifting but the prediction accuracy of the minima and maxima are comparable to the prior approach. Since the correct prediction of the states is from paramount importance for determining the optimal action, the prediction accuracy of both approaches is seen as not sufficient. Further, an erroneous prediction of c_D leads to a significant corruption of the rewards, since c_D contributes partially higher to the rewards than c_L. Therefore, in a next step it is investigated if an optimization of the model architecture may lead to an improvement in the prediction accuracy.

4.2.2 Influence of the model architecture

The current environment models used consist of 3 hidden layers with 50 neurons per layer. An optimization of the number of hidden layers and neurons per layer may improve the prediction accuracy. In order to assess the the general sensitivity of the prediction accuracy on the model architecture, a parameter study was conducted using one global environment model on the data without taking the episode number into account. The results are depicted in fig. 4.11, which shows the average L_2- and L_1-norm of prediction accuracy for predicting c_D. Although there exist some unfavorable combinations, e.g. 3 hidden layer and 25 neurons, there is no overall trend identifiable. This raises the concern, that the result of this parameter study may look completely different for e.g. a different trajectory length, making it challenging to find a combination which works well, independently of the setup.

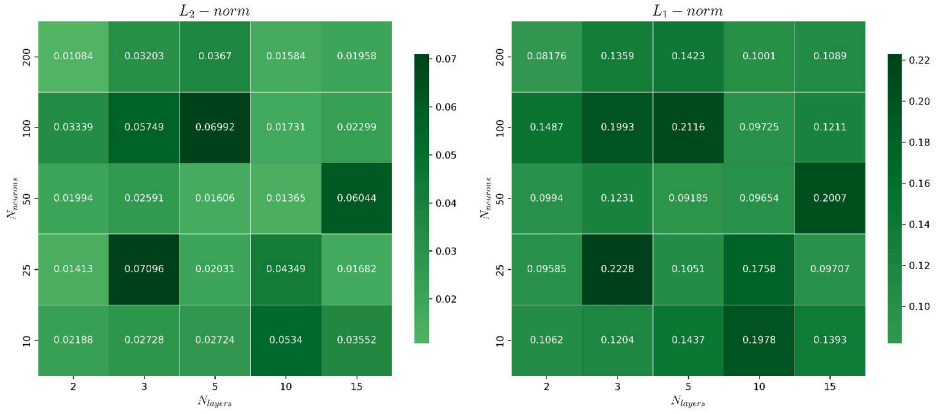


Figure 4.11: L_2- and L_1-norm of the prediction error of c_D with respect to the model architecture

The prediction accuracy for c_L, however, may be improved by optimizing the model architecture as depicted in fig. 4.12. In contrast to the results for c_D, here the prediction accuracy improves with increasing number of neurons and number of hidden layers. The L_2-norm of the error converges for $N_{neuons} \geq 25$ and $N_{layers} \geq 3$, but since the models used in all prior studies already fulfilled this requirement, it can be concluded that the potential of improving the prediction accuracy by optimizing the model architecture alone is not sufficient and the approach of modeling the CFD environment has to changed substantially.

4.2.3 Separating predictions of c_L and p_i from predicting c_D

As discussed in the previous sections, the course of the trajectories for c_L and p are similar, while the trajectories for c_D have a much more complex shape. The trajectories for c_D are generally discontinuous and show a non-periodic behavior while c_L and p behave periodically, motivating to separate the predictions of c_D from the prediction of c_L and p. This offers not only a better prediction accuracy, but also a greater flexibility since now the prediction accuracy of c_D can be improved independently without affecting the prediction accuracy of c_L and the states. Since both models require the same input and only differ in the output, the required adjustments within the implementation are only minor.

Figure 4.13a depicts the L_2- prediction error when using an episode-wise trained environment model for the states, c_L and c_D, while 4.13b shown the resulting prediction error when the

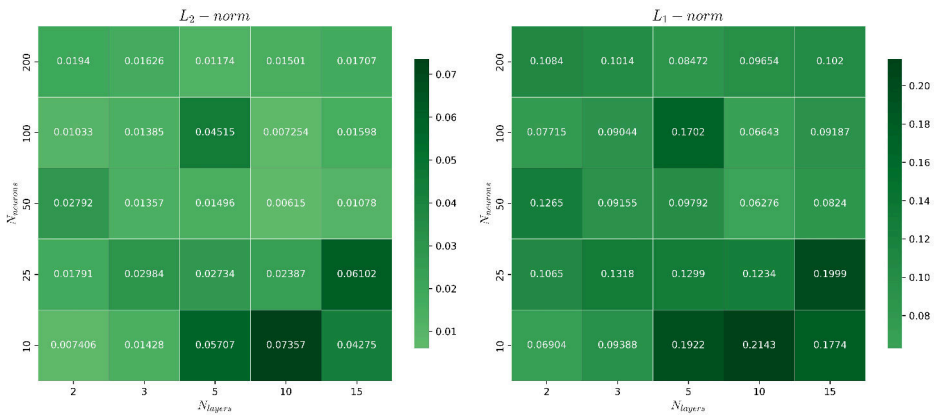
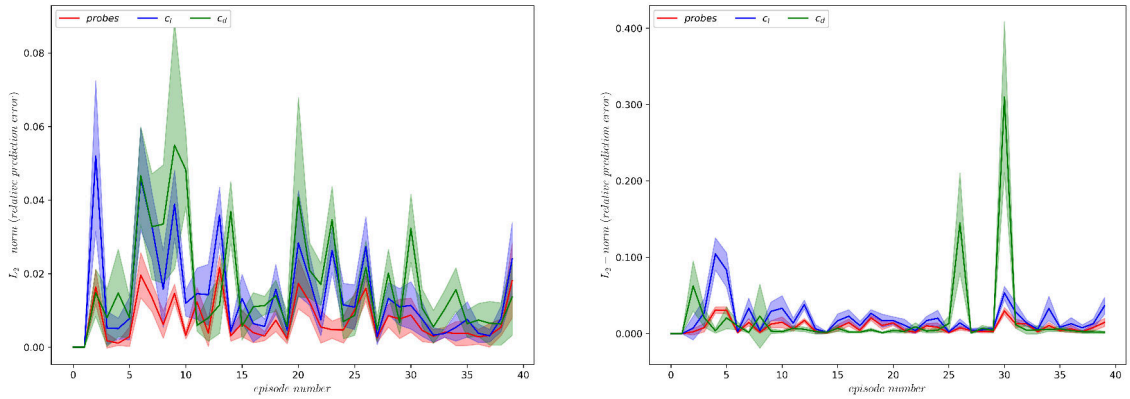


Figure 4.12: L_2- and L_1-norm of the prediction error of c_L with respect to the model architecture



(a) L_2-norm for one model predicting c_L, p_i and c_D

(b) L_2-norm for one model predicting c_L and p_i, another model for c_D

Figure 4.13: Comparison of the L_2-norm of the prediction error when using one model to predict c_L, p_i and c_D vs. one model for predicting c_L and p_i and a separate model for predicting c_D

prediction of c_D is separated form the prediction of the states and c_L. It can be seen that despite some outliers in episode 2, 26 and 30, the overall prediction accuracy improved compared to the previous approach. The prediction accuracy of the states was not affected negatively, however, the L_2 norm for c_L increased especially at the beginning of the training. Since c_L only makes up a minor part of the rewards, the correct prediction of c_D is seen as more important at this point, justifying a deteriorated prediction accuracy for c_L.

A comparison of a predicted trajectory with a real trajectory is shown in fig. 4.14. From this figure it can be seen that the prediction accuracy for c_L is in the order of the prediction accuracy of the previous methods shown. The overall course of c_L is predicted quite accurately and in contrast to prior methods, the phase-shifting problem seems to be neglectable now. The prediction accuracy of the extrema, especially the minima of c_L are worse than before. For the c_D trajectories on the other hand, separating the prediction of c_D had a great impact on the prediction accuracy. Although there is an area at episode 50...100 where the predicted trajectory diverged noticeably from the real one, the accuracy improved significantly compared to prior methods. The predictions for the states did improve as well as it can be seen in fig. 4.15. There exists still a phase-shift for some probes, which is is increasing towards the end of the trajectory. The prediction accuracy of the minima and maxima improved in general compared to the previous method.

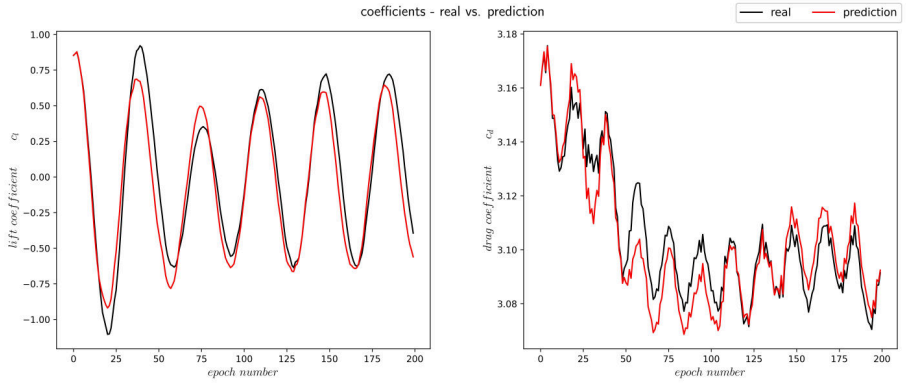


Figure 4.14: Real trajectory generated in the CFD environment in comparison to the predicted trajectory by the environment model using two models (here episode 14)

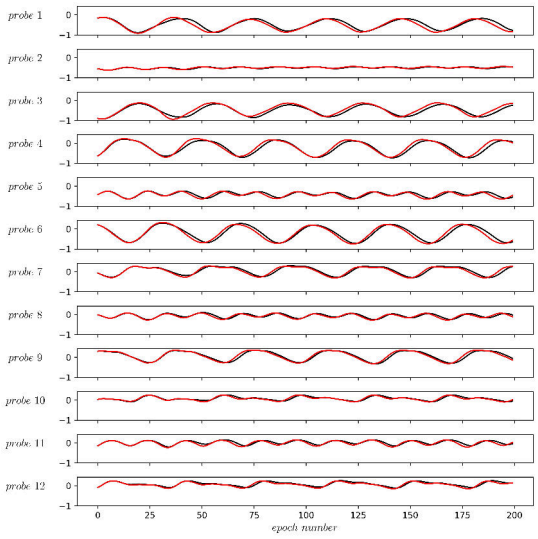


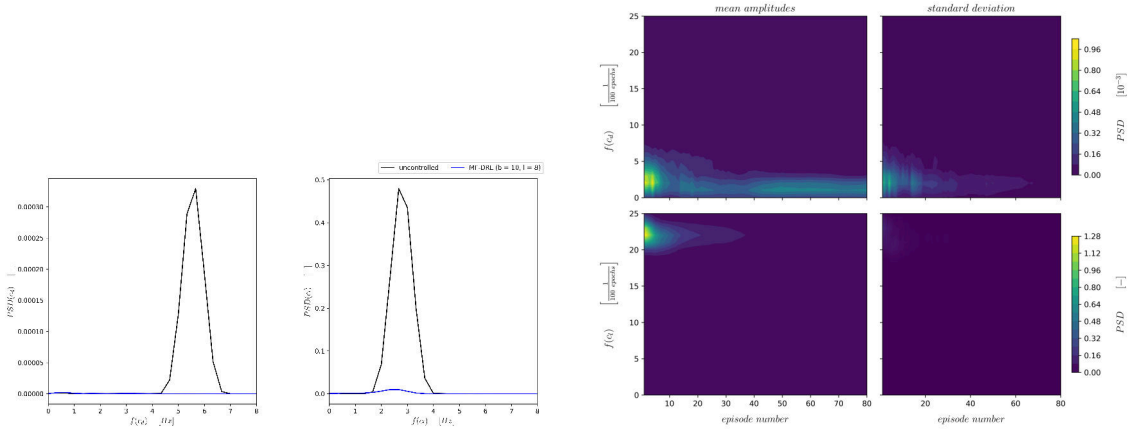
Figure 4.15: The trajectories of the probes corresponding to fig. 4.14

4.2.4 Influence of the number of input time steps on the prediction error

The prediction accuracy of the current environment model is already quite good, but especially for the c_D trajectory there are still outliers and areas where the prediction accuracy may be improved. So far, the state was predicted based on the two previous subsequent time steps. The system dynamics of the vortex shedding, however is comparably complex and underlies physical principles. A possible approach to exploit these physical principles is to tailor the number of input time steps to the characteristic vortex shedding frequencies. The existence of a possible correlation between the number of subsequent input times steps for the environment models and the frequency spectrum within the trajectories of c_L and c_D may be used in a way that the environment models simultaneously learn the underlying system dynamics rather than just predicting the next state, improving the prediction accuracy further.

In a first step, the frequency spectrum of the uncontrolled flow past a cylinder was investigated along with development of the frequency spectrum of c_L and c_D over the course of a training. The resulting power spectral densities (PSD) of the flow is depicted in fig. 4.16a and 4.16b. It can be seen that at the beginning of the training, the trajectories of c_L consist of frequencies at approximately $20...25Hz$. These frequencies are damped within the first 20...25 episodes, after which no dominant frequencies can be found in the PSD for c_L. The trajectories for c_D, however, consist of significantly lower frequency spectrum, at around $0...7Hz$ at the beginning of the training. In contrast to the frequency spectrum for c_L, the PSD of the c_D trajectories contain a larger standard deviation, which may also be a possible reason why the agent is not able to dampen the frequency spectrum significantly up to episode 80. The bandwidth of the PSD narrows to a range of $\approx 1...3Hz$ indicating a dampening behavior with progressing training, but these low frequencies are not fully dampened towards the end of the training.

The PSD of the controlled and uncontrolled flow using the final policy is shown in fig. 4.16a. Despite the frequency spectrum present during training, in the final result the frequencies for both c_L and c_D are dampened to a neglectable intensity in comparison to the uncontrolled flow. The uncontrolled flow yields one characteristic frequency of each c_L and c_D, which is in accord with the results obtained in the work of [6]. The trajectory for c_L oscillates with $f(cl) = 2.67$ Hz and an amplitude of $a(cl) = 0.479$ while c_D contains mostly a frequency of $f(cd) = 5.67$ Hz with an amplitude of $a(cd) = 0.000329$. The ratio $f(cd)/f(cl) = 2.125$ indicates that the periodic length of c_L can be seen as the decisive parameter, since it requires substantially more input time steps to cover one period of c_L. Considering the sample frequency of $100Hz$, the results



(a) frequency spectrum of the final policies of the controlled and uncontrolled case

(b) Power spectral density of c_L and c_D throughout the training

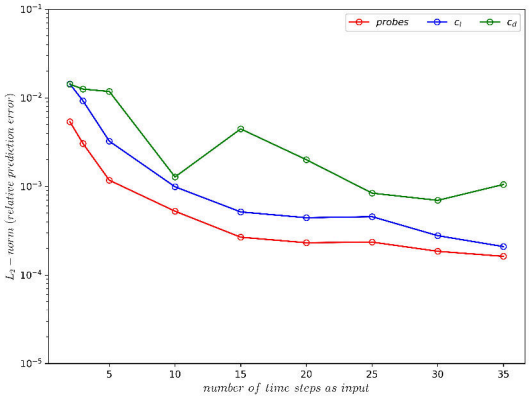
Figure 4.16: Power spectral density of the trajectories for c_L and c_D for the controlled and uncontrolled flow

of this frequency analysis suggests approximately 30...40 time steps as model input in order to cover one period with a frequency of $f(c_L) \approx 2.5...3Hz$.

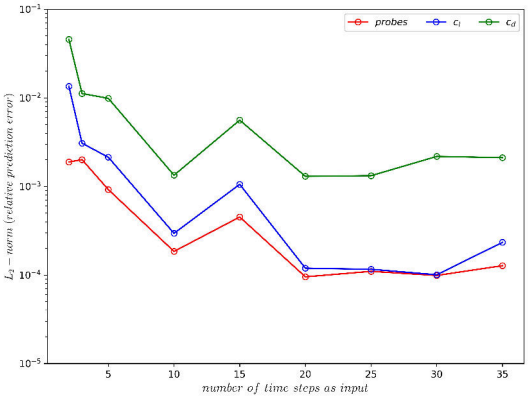
The number of input time steps can not be chosen arbitrarily high, since there is only a limited amount of data for training-, validating and sampling the initial states from; in the present case the previous two episodes. The more input time steps are required, the less training data is available, due to the fact that the training data can not be used for sampling the initial states from, which may lead to more inaccurate models. Another aspect is the variance within the sampled initial states. The initial states are sampled from the CFD data, which was already seen by the agent and is therefore not contributing anymore to the overall training process of the agent. With increasing number of input time steps, the relative amount of new interactions between the environment model and agent is decreasing, because the trajectory length is kept constant independent of the amount of time steps used as input. Additionally, the combination of a limited amount of data to sample initial states from and large buffer sizes on the other hand may lead to the necessity of re-using data points for initial states, causing the initial states to repeat. Since the policy network is not updated within one episode, two trajectories with the same initial state are redundant and lead to the exact same results since the models predict the same output when the same input and action is given. As a consequence of these issues, the goal is to keep the number of required input time steps to a minimum while maximizing the prediction accuracy of the environment models.

In a first step, the usage of one and two global models respectively is re-visited as comparison to episode-wise trained models. Figure 4.17a presents the average L_2-norm of the prediction error when one global model is used while fig. 4.17b shows the prediction error when c_D is predicted separately. It can be seen that for both cases, the prediction error decreases overall with increasing number of time steps, hence the models tend to be more accurate. However, this behavior is partially caused by the constant trajectory length, leading to a decrease in the actual amount of points predicted, when increasing the number of input time steps. For a trajectory length of $l = 2s$ for example, when using 2 subsequent time steps as input there are 198 time steps remaining which may all contribute to the L_2 prediction error. If the number of input time steps is increased to e.g. 30 this leaves only 170 time steps for prediction and potential prediction errors.

Another trend which can be observed is the decoupling of the prediction error for c_D when two environment models are utilized. From fig. 4.17a it can be seen that cl, p and c_D decrease



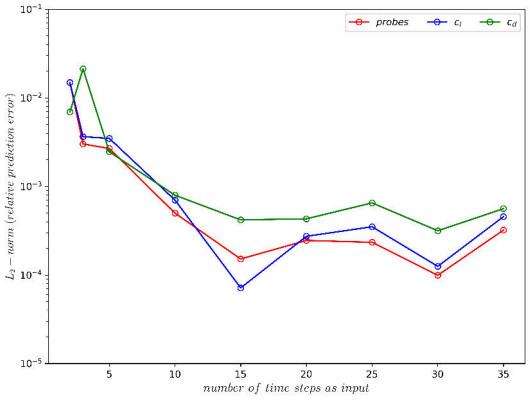
(a) L_2-norm one global model



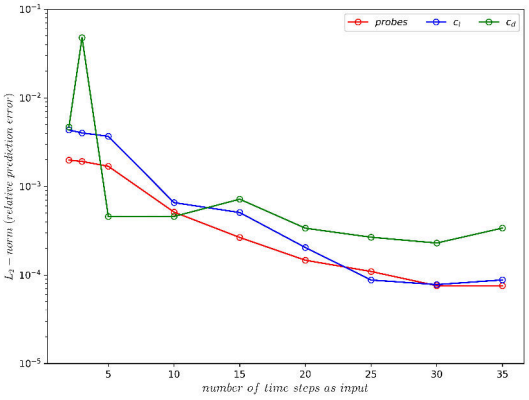
(b) L_2-norm two global models

Figure 4.17: L_2-norm of the prediction error depended on the number of time steps used as model input. The environment model(s) are trained independently of the episode number.

similarly with increasing number of input time steps. When separating the predictions for c_D as it is shown in fig. 4.17b, the prediction accuracy of c_D diverges from the error of c_L and p starting at $N_{t,input} = 20$. This behavior can be mitigated when training the environment models episode-wisely as it is depicted in fig. 4.18a and 4.18b. In contrast to the globally trained environment models the prediction error when training the models episode-wise is almost decreased by one order of magnitude for the prediction of c_D, emphasizing the advantages of this approach. When separating the predictions for c_D, the prediction accuracy of c_D remains nearly the same as if one model is used, but this result may change depending on the current episode (the here shown plots are for episode 40) and may also be a consequence of a sub-optimal network architecture of the c_D model. The prediction error for c_L and p showed here a minor decrease compared to utilizing one environment model for $N_{t,input} \geq 20$. Further, there exists a convergence behavior for the predictions when $N_{t,input} \geq 15$ and one environment model is used while the prediction error converges $N_{t,input} \geq 20...25$ if two environment models are utilized for the predictions. This indicates that it is not required to display one complete period within the input data, despite the approach used.

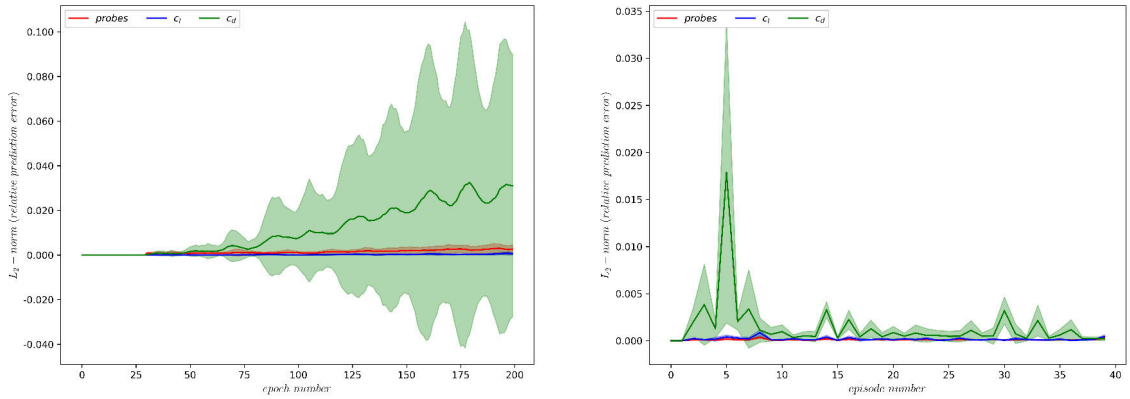


(a) L_2-norm one episode-wise model



(b) L_2-norm two episode-wise models

Figure 4.18: L_2-norm of the prediction error depended on the number of time steps used as model input. The environment model(s) are trained with two CFD episodes and then tested on data of the following episode. The L_2 error is therefore the prediction error of the test data, in this case for episode 40.



(a) L_2-norm two global models **(b)** L_2-norm two episode-wise models

Figure 4.19: L_2-norm of the prediction error for $N_{t,input} = 30$ with respect to the epoch number

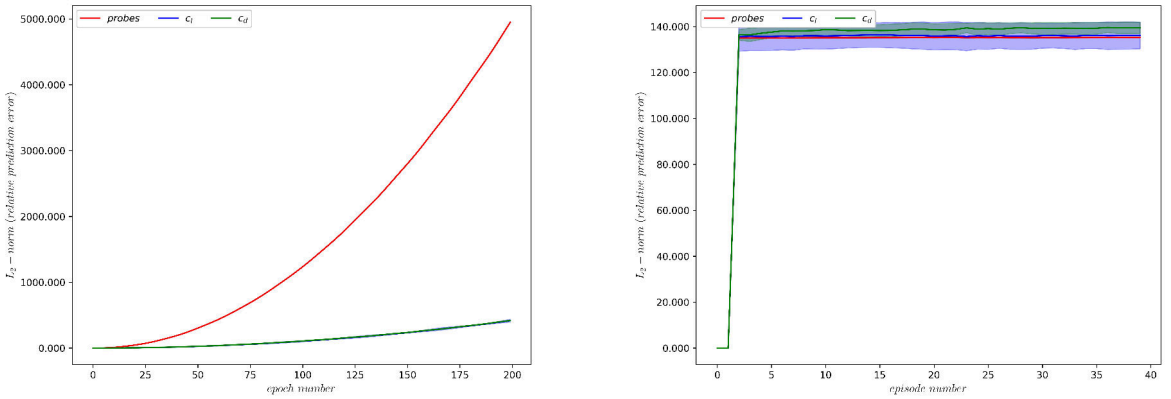
Finally, fig. 4.19a shows the prediction error when two global environment models and 30 time steps as input are used. Clearly, the prediction accuracy of c_D is significantly worse than for c_L and p additionally confirming that a globally trained model is not capable of capturing the changes within the trajectories over the course of the training. The prediction error when using two models trained for a specific episode is on the other hand approximately two orders of magnitudes smaller when neglecting outliers as depicted in fig. 4.19b. As a consequence of the aforementioned optimization problem between prediction accuracy and variance within the data for training the models and sampling initial states, in the following the usage of 30 subsequent time steps have been found to be a good compromise. Further, a dependency of the number of input time steps on the frequency spectrum could not have been observed in general, which decreases the importance of the choice of $N_{t,input}$ on the model accuracy.

4.2.5 Low-pass filtering of the c_D trajectories

The prediction accuracy of the c_D-trajectories is generally about one order worse than for e.g. c_L, despite the number of time steps used a model-input. In contrast to the c_L- or p_i-trajectories, the c_D-trajectories contain discontinuities and a stronger non-periodic behavior. This motivated the application of a low-pass filter for the c_D-trajectories in order to improve the prediction accuracy. However, a low-pass filtering had no measurable impact on the prediction accuracy for the c_D-model and was therefore not regarded further. There was additionally a concern that a low-pass filtering of c_D alters the rewards, but since the states remain the same this behavior may increase the model-bias.

4.2.6 Prediction of the change of state instead of the next state

A common practice presented in literature about DRL algorithms is the prediction of the change of state instead of the next state itself. This approach, however, was found to produce complete nonphysical results as shown in fig. 4.20a and 4.20b. In both cases, namely one global environment model as well as when training one model each new episode, the prediction error was unrealistically high in the order of $10^2...10^3$. This behavior occurred despite the number of models trained or number input time steps used. However, the prediction of c_D improved when using a separate model for predicting c_D as shown in fig. 4.21. Since the implementation for predicting c_L and p_i was the same as for predicting c_D, and the predicting of c_D is at least in the same order of the real trajectory; an implementation error could have been ruled out as possible cause of this unrealistic behavior.



(a) L_2-norm training one global model

(b) L_2-norm training new models every episode

Figure 4.20: L_2-norm of the prediction error when predicting the change of state instead of the new state (here: *ds* was scaled to interval $[0, 1]$)

Further, when scaling the model input to an interval, e.g. $[0, 1]$, as it is common practice; in contrast to all prior findings the prediction error increased significantly up to values in the order of $10^{14}...10^{17}$. A cause for this unusual behavior could not have been determined, but is may be a result of round-off errors due to the fact that the change within the state is small. In order to predict the next change within the state, the current state needs to be computed by an addition (or subtraction) of the previous state and the predicted change in state corresponding to the previous state. This addition (or subtraction) of a value in the order of 1 to a value with a significantly lower order followed by a scaling of this value may be prone to round off errors resulting in a high prediction error.

As a consequence of this behavior, the usage of the change of state was completely discarded for all further implementations. For an integration of a model-based training into the PPO-training routine, two environment models where chosen, each model takes 30 subsequent time steps as input. The models itself are trained with CFD data of the current episode, further details of the integration into *drlfoam* will be discussed in the next section.

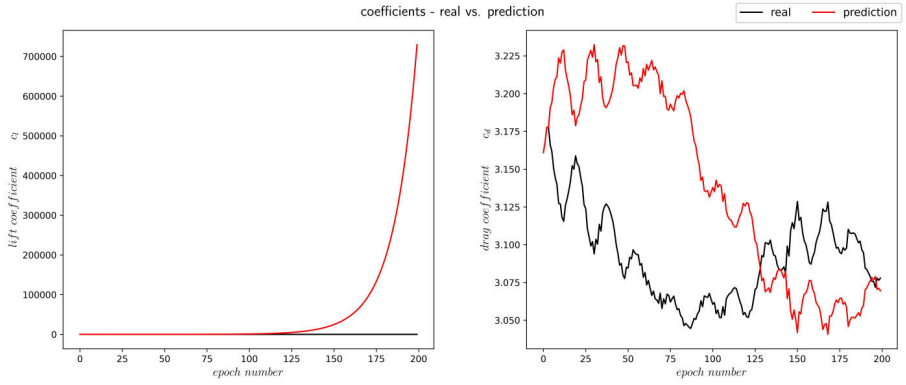


Figure 4.21: Real trajectory generated in the CFD environment in comparison to the predicted trajectory, in this case *ds* was not scaled to $[0, 1]$ since it has been found that a scaling to $[0, 1]$ deteriorates the prediction accuracy even further (here episode 14)

Chapter 5

Results

In this chapter, the result of integrating the environment models into the PPO-training routine will be presented and discussed. The approach presented at the end of the previous section is used as a starting point. This approach, however, caused new issues when integrated into the PPO-algorithm, such as the lack of training data as a consequence of diverged CFD simulations. These issues were solved successively by modifying the training routine until a fairly stable routine could have been devolved yielding results comparable to the model-free training. At the end of this chapter, the modified approach of approximating the CFD environment with deep neural networks was evaluated in comparison to the model-free counterpart for different Reynolds numbers.

5.1 Initial approach of integrating a MB-training into *drlfoam*

The CFD environment was modeled using one model for predicting c_L and p_i and another model for predicting c_D in an initial approach. Each model utilizes 30 subsequent time steps as input, the trajectories are initialized by randomly sampling 30 time steps out of the trajectories generated within the CFD environment throughout the current episode. Since this initial approach was meant to proof the overall feasibility of an integration into the PPO-training routine of *drlfoam*, all trainings are run for only 50 episodes and not averaged over multiple different seed values. This was done in order to minimize the computational costs and is not representing the common practice, however, it ensures the ability to conduct a variety of parameter studies and tests in order to find all possible sources of errors that occur as a consequence of integrating the environment models into the PPO-training routine. The trajectories of the model-based episodes were generated in the following using one model for predicting c_D and another model for predicting the states and c_L.

The models are initially trained in episode zero, after the CFD simulation finished and prior the policy network is updated. In the following episodes, the buffer is filled with trajectories generated by the models until an episode $e\%5 = 0$ is reached. For all further episodes $e\%5 = 0$, the training switches back to CFD, meaning the trajectories are generated within the CFD environment. These trajectories are then used to train the next environment models. The reason for this choice was justified by the observation that the model training is equivalent to approximately 1...3 CFD episodes with respect to the runtime, depending on the trajectory length. To make the MB-approach competitive to the MF-training with respect to the required runtime, at least three consecutive MB-episodes have to be conducted. When considering a safety margin of one episode due to varying available resources on different systems, this leads to four consecutive MB-episodes followed by one episode in the CFD environment. The reference case with respect to the rewards is a MF-training with a buffer size of $b = 8$ and trajectory length of

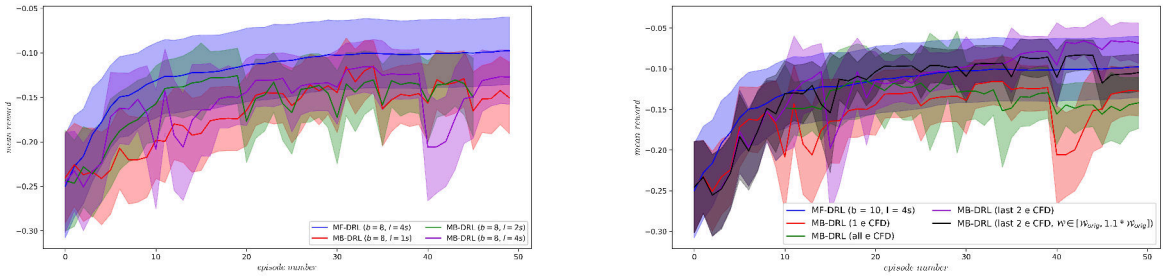
$l = 4s$, the model training itself was conducted using 4 CPU's in parallel. The hyperparameters of the environment models remain the same and can be found in the appendix, section A.5.2.

5.1.1 Optimizing the number of episodes for training the environment models

The environment models are trained in a first step only with the data of the current CFD episode. It is important to note, that throughout this thesis the models are generally trained with trajectories generated within the CFD environment, although it is possible to train the models with the model-generated trajectories. This idea was discarded due to model inaccuracies raising the concern of a destabilization of the training routine and further deterioration of the model accuracy and accompanied exploitation by the policy. The usage of CFD data for model training assures that the relation of states, actions and rewards display the correct system dynamics.

Figure 5.1a displays the rewards received for a MB-training in comparison to the MF-counterpart. It can be seen that the rewards of the MB-trainings yield significantly worse results, independent of the trajectory length. The course of the rewards is further highly unstable and discontinuous compared to the MF-training, indicating that the environment models are not able to predict the next state sufficiently accurate. These prediction errors may lead to the observed trend that the trajectory length is not affecting the rewards in any measurable way. A possible explanation of this behavior with respect to model inaccuracies is the fact that only the current CFD episode is used for model-training. This may lead to a so-called *catastrophic interference* or *catastrophic forgetting* [7]. Catastrophic interference refers to the problem, that the models are updated using the latest data, which causes the model to forget the data already seen in prior updates to a certain degree. This directly increases the epistemic uncertainties, since the amount of training data is not sufficient to cover all possible states. In the context of CFD simulation, this is especially an issue since the generation of data with CFD is computationally demanding and consequently costly. In order to mitigate the effects of catastrophic interference and to improve the sample efficiency of the data, it may be beneficial to not only use the current CFD episode for training, but also the previous CFD episodes, which may lead to a more robust policy and consequently more stable learning.

The influence of number of CFD episodes used for model training is depicted in fig. 5.1b, the meaning of the case with additional entropy will be explained later. Firstly, it can be seen that if the training data is extended to the current and previous CFD episode, the rewards improve significantly, and are even able to outperform the results of the MF-training. Additionally, the course of the rewards is stabilized and does not show as many sudden decreases of the rewards compared to the training with only the current CFD episode. A further increase of the number of CFD episodes utilized, however, leads to a deterioration of the rewards, as indicated by the



(a) only current CFD episode used for training **(b)** multiple CFD episodes used for training

Figure 5.1: Different approaches for training the environment models, all cases run with $b = 8$ and $l = 4$. Parameter e denotes the *episode* number while $\mathcal{W}$ denotes the additional entropy with respect to the original entropy $\mathcal{W}_{orig}$

green curve in fig. 5.1b. In this case, all CFD episodes encountered up to the current episode are used for model-training. A possible reason for this trend may be a generalization of the environment models due to the high variance within the training data as already could have been observed in section 4.1, leading to an overall decrease in model accuracy and consequently rewards. When the trajectories are not homogeneously, the environment model is not able to learn the system dynamics sufficiently accurate, because especially at the beginning of the PPO-training the agent is exploring the parameter space using random actions. An advantage of increasing the number of CFD episodes used may be an increase in stability of the training routine as a consequence of a decrease in epistemic uncertainties caused by a larger amount of data, which can be concluded by the smoother course of the rewards depicted in fig. 5.1b. This increase of data leads to a greater variety of different states seen during the training process by the models and although the prediction accuracy is decreased compared to training where the models are only trained with the current CFD episode, the models are more accurate on predicting unseen states. Further, the total number of possible unseen states decreases as a direct consequence of the larger amount of training data. The number of episodes chosen for training the environment models can therefore be seen as an optimization problem of stability on the one hand and performance in terms of rewards, on the other hand.

To emphasize the change within the variance of the trajectories throughout the PPO-training, fig. 5.2 depicts the variance of the beta-distribution of the policy network with respect to the episode number. The variance of the beta-distribution is measure for the uncertainty of the chosen action by the policy network, since only the mean value of the beta-distribution is used to compute the action. A high standard deviation consequently indicates a high uncertainty of the policy network that the chosen action is the optimal one. From fig. 5.2, it can be seen that the variance decreases throughout the MF-training from ≈ 0.058 in episode zero to ≈ 0.01 for episode 50 and is likely to decrease further for higher episode numbers. This behavior was expected since at the beginning of the training the entropy is high in order to force the agent to explore the parameter space, while towards the end of the training an exploitation is preferred over exploration. For the MB-trainings, however, it can be seen that the variance decreases significantly faster than for a MF-training and converges at lower values. This behavior indicates that for a MB-training the policy network is certain that the chosen action is the optimal one, although this might not be the case as indicated by the peaks when running an episode in the CFD environment. The differences within the uncertainty of MB- and MF- episodes may be a consequence of the poor prediction accuracy of the environment models, leading to exploitation of these prediction errors by the agent, as discussed in section 2.3.1.

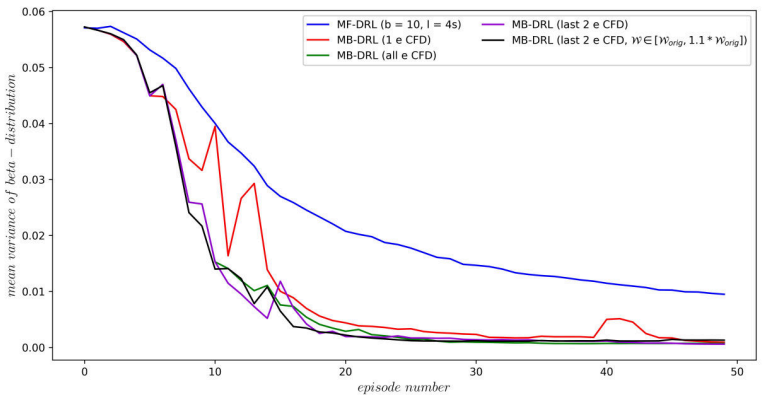


Figure 5.2: Corresponding variances of the Beta-distribution with respect to the training routine

To mitigate this problem, the effect of introducing additional entropy to the MB-training was investigated. The entropy was therefore artificially increased using a beta distribution. At the beginning of the training, the entropy and consequently the uncertainty is already sufficiently high while towards the end of the training the entropy should decrease as a consequence of reducing the exploration of the agent. The beta-distribution was utilized, since it is zero at both boundaries while the maximum value of the additional entropy can be easily controlled by the two parameters α and β. The beta-distribution for generating additional entropy can be found in the appendix, section A.4 along with the parameters α and β.

Fig. 5.1b shows the resulting rewards when introducing additional entropy to the MB-training. It can be seen that, despite stabilizing the course of the rewards to some degree, it did not yield any improvement compared to the case without additional entropy. Further, this approach was not able to increase the resulting uncertainty of the policy network, as indicated by fig. 5.2. A possible explanation for this behavior might be that additional entropy is operating as noise within the data. Noisy training data is generally known the deteriorate the model accuracy leading to a worse performance, e.g. as presented in [5]. Consequently, this idea of introducing additional entropy was not considered further, however, extending the amount of training data from only using the current CFD episode to using the current and the previous CFD episode was found to have a great beneficial effect of the training performance and is therefore used in all following trainings.

5.1.2 Influence of the trajectory length

The previous section showed that the MB-training routine was able to run for 50 episodes and yielded partially comparable results to the MF-training routine. However, especially at the last CFD episode the training crashed in some cases indicating stability issues when increasing the number of episodes. In order to assess the long-term stability of the model-based training, the number of episodes was increased to $N_{episodes} = 80$ and the results where averaged over three different seed values in order to account for the dependency of the rewards on the initialization. During model-training no additional entropy was introduced, but in contrast to the results presented in the previous section, in the following the trajectories of the current and last CFD episode was used for training the environment models. Additionally to the training stability, it was investigated if this modification leads to improved rewards with respect to the trajectory length, since the initial approach for the MB-training showed no dependencies of the trajectory length on the received rewards. The reference case remains with $l = 4s$ and $b = 10$ the same as used in the previous section.

General remarks for all remaining trainings presented in this chapter

Due to stability issues occurred during a MB-training, some of the conducted trainings crashed. This crash was generally caused by a divergence of the CFD simulation leading to lack of training data. Although this problem could have been mitigated, as will be presented in the next sections, it could not have been solved completely. To ensure the comparability of the results to each other and to the corresponding MF-training, the trainings where re-started with different seed values, until three completed trainings were available for each setup; consequently, the seed values may not be the same for each case. Further, as will be discussed in the following sections as well is the dependency of the seed value on the system where the training is conducted. These problems may lead to the unfavorable situation, that the presented results in this thesis may be not fully reproducible quantitatively. However, for the sake of completeness it is important to mention that the initialization was started with $seed = 0$, then the seed value is incremented by one until three fully converged training where available for each case. In some cases, the stability of the training was not sufficient to even reach three converged trainings, in those cases the training was averaged only over all available seed values. This fact will be pointed out in the

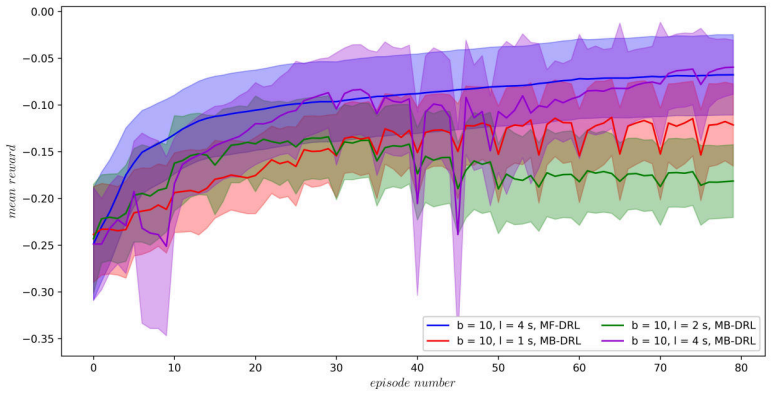


Figure 5.3: Influence of the trajectory length on the performance for a model-based PPO-training

following, whenever it was not possible to average over three different seed values. Theses pieces of information and procedures apply for all trainings presented over the course of the following chapters.

Figure 5.3 shows the resulting rewards received in the MB-training with different trajectory lengths. In contrast to the results presented in the previous section, the course of the received rewards show a dependency on the trajectory length. But while the rewards for a MF-training increase monotonically with increasing trajectory length, when running a MB-training this seems not to be the case. Here, a trajectory length of $l = 1s$ yields significantly higher rewards than the training with a trajectory length of $l = 2s$, a reason for this behavior could not been found. Further, the rewards for a training with $l = 1s$ and $l = 2s$ did not show any sudden decreases throughout the course of the training. This confirms the assumption that using two consecutive CFD episodes for model training improves the stability of the model-based PPO-training, but the training stability in general is still not nearly as high as for the MF-trainings.

Another difference between MF- and MB-training can be found in the resulting c_L and c_D values with respect to the episode number. As can be seen in fig. 5.4, the course of the c_D values matches the trend displayed by the rewards, since c_D has a large influence on the reward. The c_L values of the other hand can be dampened to values close to zero in MF-training. For a MB-training, the c_L values remain with values up to $|c_L| \approx 0.2$ significantly higher. However, since c_L is only weighted with 0.1 in the reward function, the divergence of c_L has only a neglectable influence on the rewards.

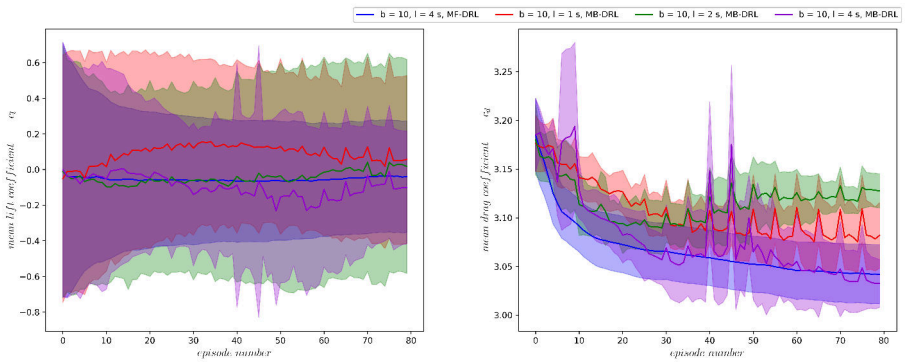


Figure 5.4: c_L and c_D over the course of the MB-PPO training when using different trajectory lengths

Since there could not been found a reasonable explanation for this behavior, in a next step the performance of the final policy is evaluated. The rewards of the MB-training for $l = 4s$ are comparable to the MF-counterpart, therefore these two cases were compared to each other in the following section aiming to find potential causes for the stability issue, which occurred during the MB-training.

5.1.3 Assessment of the final policies

In order to assess the final policies, the seed value yielding the best rewards was taken. For the model-free training, the policy of episode 80 was used for controlling the flow while for the MB-case the policy of the last CFD episode, namely episode 75 was taken. This ensures the comparability of the policies since the rewards of the MB-episodes are generally higher than the rewards received in episodes run in the CFD environment. Figure 5.5 shows the course of c_L and c_D using the final policies to control the flow. It can be seen that after the control starts at $t = 4s$, it takes approximately $1s$ to reduce c_L and c_D to a level with low values, as already observed in section 4.1. After this initial, transient phase the agent is able to maintain c_L and especially c_D at a low level, while the performance of the MB-policy yields comparable values as the MF-training, which is also presented in table 5.1. This shows the general feasibility of MB-DRL for flow control problems, despite the decreased stability of a MB-training compared to the MF-counterpart.

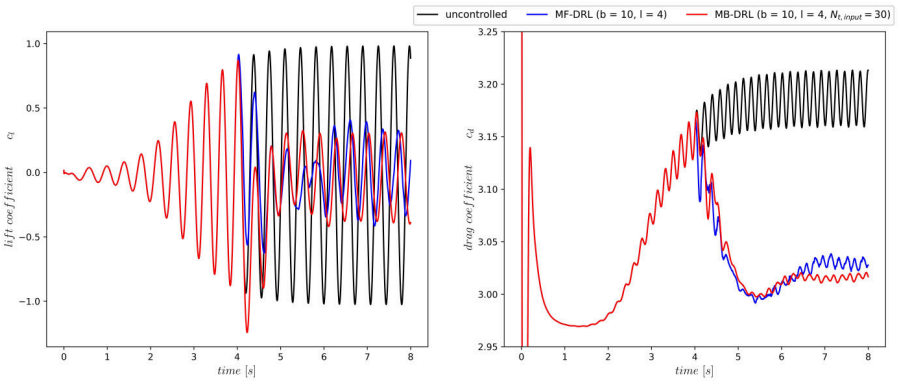


Figure 5.5: Results using best policy for MF case and last CFD episode for MB case

Since the performances of the final policies are comparable to each other with respect to the load reduction, this raises the question of what may cause the stability issues during a MB-training. A possible reason may be the quality of the model-generated trajectory throughout the training as depicted in fig. 5.6. It is important to note, that the here shown trajectories are not meant to have the same values, since they originated from different trainings, therefore, the goal is to emphasize the qualitative differences between the model-generated trajectories and trajectories generated within the CFD environment in a MF-training. As depicted in fig. 5.6, the course of the trajectory for c_L is qualitatively quite similar although the absolute values for c_L of the MB-training are generally worse compared to the MF-training. For the trajectory of c_D, however, it can be seen that the trajectory generated by the environment model shows, in contrast to

case	$\mu(c_D)$	$\mu(c_L)$	$\sigma(c_D)$	$\sigma(c_L)$
uncontrolled	3.1863	−0.0106	0.0186	0.7119
MF-DRL	3.0257	0.0032	0.0075	0.2461
MB-DRL	3.0157	−0.0680	0.0026	0.2441

Table 5.1: Average c_L and c_D values of the final policies in the quasi-steady state at $t = 6...8s$, both controlled cases where run with $b = 10$ and $l = 4s$

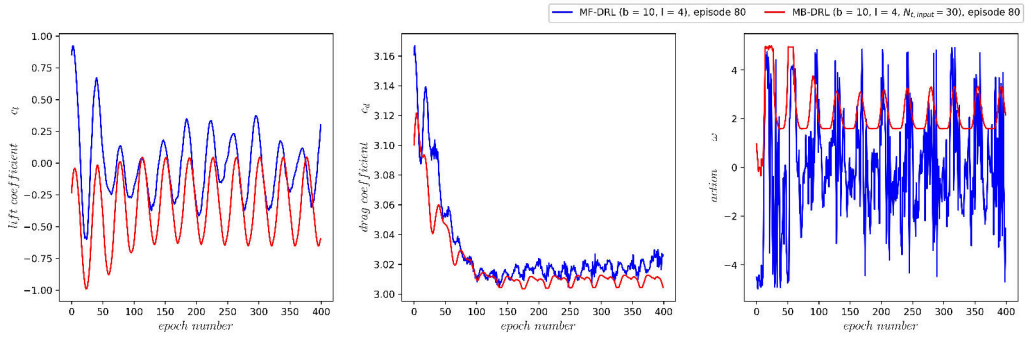


Figure 5.6: Qualitative comparison of the model-generated trajectory with a trajectory generated within the CFD environment, both for episode 80

the trajectory of the MF-training, a distinct periodic behavior and a less discontinuous course. This can be seen as a direct consequence of the actions taken over the course of the trajectory, which is depicted on the right of fig. 5.6. The actions for the MB-trajectory show a remarkable periodicity at significantly lower frequencies than the MF-counterpart, also the actions are all positive. This behavior implies that the environment models act as a low-pass filter, especially for ω and are therefore not capable of covering the full frequency spectrum.

In order to analyze this issue further, the PSD of the actions is shown in fig. 5.7 along with the frequencies and their multiples of c_L and c_D. It can be seen that for a model-free training, the actions oscillate approximately with the frequencies of c_L and c_D and multiples thereof, while the frequencies of c_L are the dominating factor. This is caused by the fact that the frequencies of c_L are in the range of the frequencies of the states, which can be found in the appendix, fig. A.9. The PSD for a MB-training, however, shows that the actions are not able to display frequencies greater than two times of the frequencies of c_L and c_D confirming that the environment models act as a low pass filter with a cut-off frequency of approximately $6Hz$.

In a next step, it was investigated if the number of input time steps has an effect on the low-pass filtering behavior of the environment models. The results of the PSD for different $N_{t,input}$ are shown in fig. 5.8. It can be seen that for $N_{t,input} = 15$ and $N_{t,input} = 60$ there exist small peaks in the frequency spectrum at frequencies corresponding to 2 and 3 times of the frequencies of c_D. These peaks, however, have an insufficiently low amplitude to actually influence the periodic behavior of the actions. In all cases, the environment models seem to act as a low-pass filter on the actions, which indicates that the number of input time steps can be ruled out to be the reason for this behavior, as indicted by fig 5.8.

A possible explanation for the low-pass filtering of c_D and actions may be that only the trajectories of the probes are used as input into the policy network. In a CFD environment, there

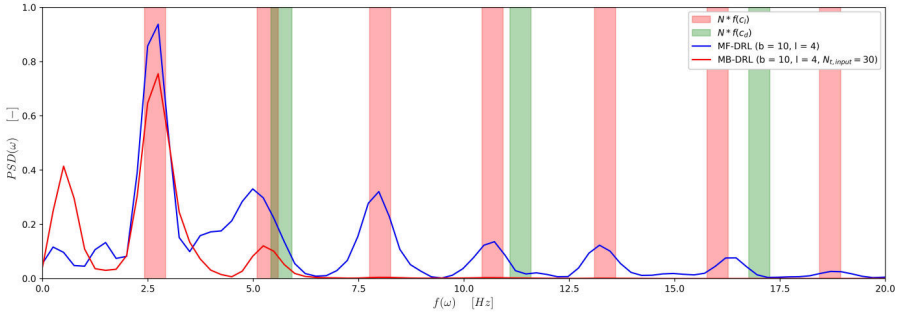


Figure 5.7: Power spectral density for ω and frequencies of c_L and c_D

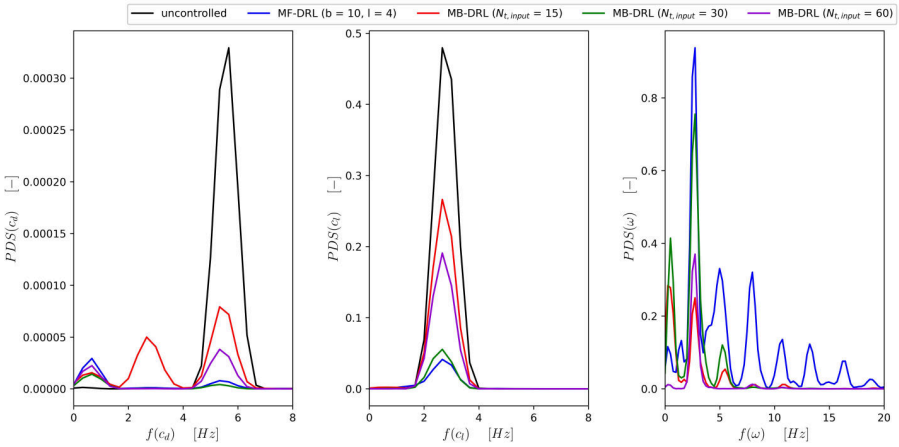


Figure 5.8: Power spectral density for c_L, c_D and ω with respect to $N_{t,input}$

exists a physically dependency between the pressure in the wake, and the c_L and c_D values. The environment models, however, are not based on physics and consequently this dependency is not naturally given. The probes oscillate with frequencies in the range of the frequencies of c_L. This may cause the models to encounter periodic input values, which oscillate in exactly those frequencies. If the input values are similar, it is likely that this leads to a similar output. The policy network samples its actions depending on the current state, consequently a periodic behavior of the model output leads to periodic actions, since the agent is not updated during the generation of the trajectories. In order to avoid this behavior, a possible approach may be the introduction of discontinuities into the trajectories, which can be achieved by using an model-ensemble (ME) instead of only one model as a consequence of the aleatoric uncertainties. Since every model within the ensemble is trained differently, each model will predict a slightly different state given the same actions and input values, as e.g. shown in [4]. If a model is randomly chosen out of the ensemble every new time step to predict the next state, this is likely to artificially create discontinuities leading to higher frequencies within the trajectories. The extension to a model-ensemble will be presented and discussed in the next section.

5.1.4 Extension to model-ensemble

The following section discusses the influence of using a model-ensemble on the results and stability of the MB-training. In all further cases, the number of models always refers to the number of models predicting c_L, p_i and c_D, e.g. $N_{models} = 1$ means that there is one model for predicting c_L and p_i and another model for predicting c_D; all models in the ensemble are trained for 10 000 epochs. The number of input time steps as well as the model architecture remains the same as in the previous section. However, there have been made some modifications regarding the training routine which shall be presented briefly.

The optimizer was changed from *Adam* to *AdamW* due to its improved convergence behavior. Further, an early stopping criteria was introduced in order to accelerate the model training. The early stopping was based on the validation loss, if the validation loss decreases to values lower than 10^{-5} the model-training was stopped. Additionally, the exception handling and consequently stability of the training routine was improved in case CFD simulations diverged during the PPO-training. If the buffer of the current episode returns empty trajectories, meaning all or at least some CFD simulations diverged, then the trajectories three CFD episodes are loaded and used additionally to the CFD data of the previous CFD episode. This ensures that the available data for model-training and sampling initial states is always kept at two times the buffer size. In case two subsequent CFD episodes return an empty buffer, then only the last converged CFD episode is utilized for model-training. A potential issue of using trajectories

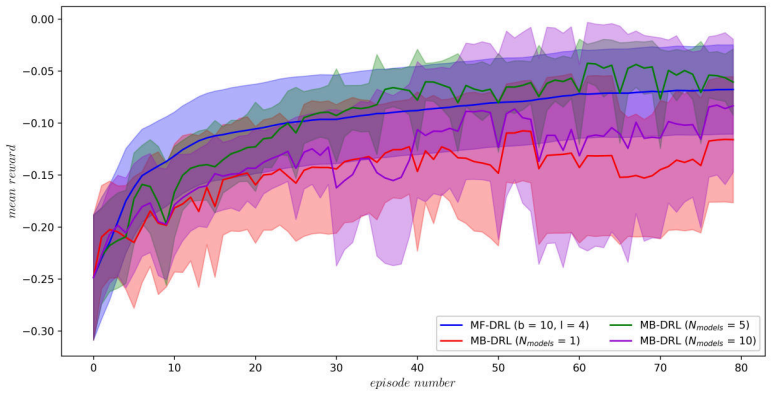


Figure 5.9: Rewards received for an MB-training with $b = 10, l = 4$ and different N_{models} in comparison to the MF-training

generated too far in the past is the divergence of the policy used to generate the data and the current policy. As a consequence, if the policy under which the training data was generated and the policy used for making the predictions are too different, the model-generated trajectories are likely to be highly inaccurate. This leads to a deterioration of the current policy from which the training can not recover. It was therefore decided to abort the PPO-training if there are no converged CFD simulations within three consecutive CFD episodes, which corresponds to 15 episodes in total.

The rewards received for trainings with varying amount of environment models in the ensemble is shown in fig. 5.9 for a trajectory length of $l = 4s$. The results for $l = 2s$ and $l = 6s$ show qualitatively the same trend and can be found in the appendix, fig. A.10 and fig. A.11. When increasing the number of models from one to $N_{models} = 5$, the rewards increase significantly, but at the same time the differences between rewards received in a model-based episode are diverging from the rewards received within the CFD environment. An increase to $N_{models} = 10$, however leads to a deterioration within the rewards down to a level comparable to the performance of $N_{models} = 1$. The standard deviation of the training with $N_{models} = 1$ is significantly increased compared to the results presented in the previous section, probably resulting from a poor convergence behavior of the CFD simulations and the associated modifications made with respect to the loading of the trajectories for model training. Before a possible reason for this decrease in performance is given, the divergence of the rewards as well as the problem of low-pass filtering shall be discussed.

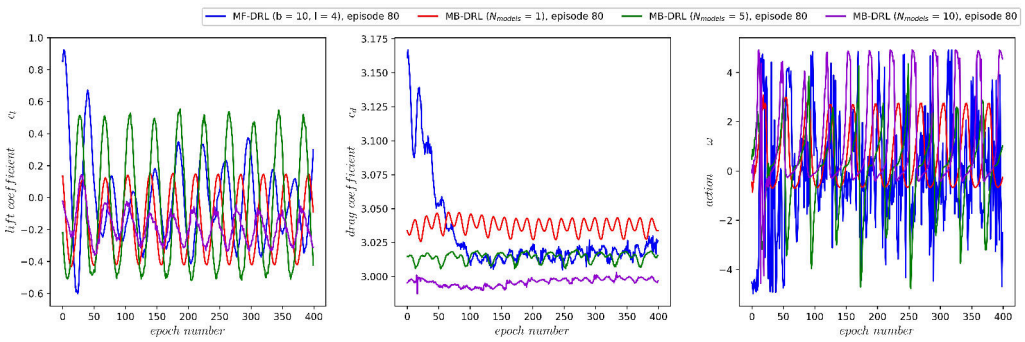


Figure 5.10: Qualitative comparison of the trajectories for c_L and c_D at episode 80, for $b = 10, l = 4$ and different N_{models} in comparison to the MF-training

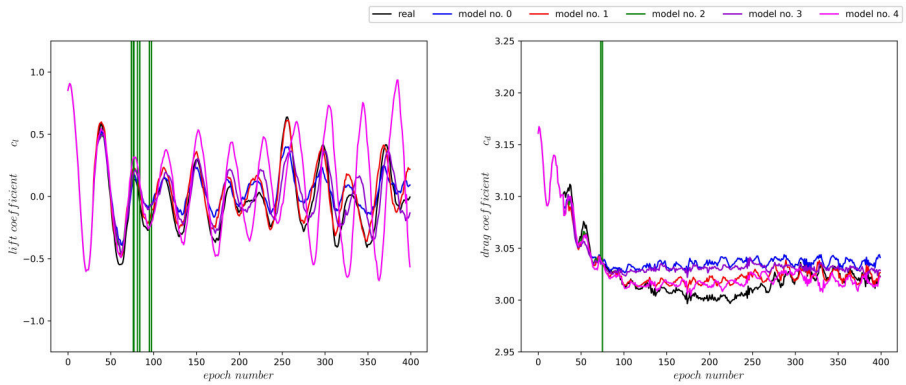


Figure 5.11: Prediction accuracy of the different models for episode 80, $l = 4s$ and $b = 10$

Figure 5.10 shows trajectories randomly drawn from episode 80. Again, since these trajectories are taken from different trainings, they are not meant to yield the same values. However, it can clearly be seen that while for the MF-training the trajectory for c_D starts with rather high values, all MB-trajectories start with significantly lower values for c_D. This is caused by the sampling of the initial states for generating the MB-trajectories. In order to provide 30 coherent time steps for the model input, the initial states are randomly sampled from trajectories generated within the CFD environment. This procedure was implemented in order to provide a sufficient amount of initial states, since the amount of available trajectories for model-training and sampling is strongly limited. That means that the initial transient phase of those MF-trajectories, starting at the uncontrolled flow and making up approximately the first second of the trajectory, is not necessarily sampled as initial state. This transient phase yields in general higher c_D values than, e.g. the last 30 time steps of the trajectory, leading to a higher average c_D value for the complete trajectory. Since the c_D value contributes the major part of the rewards, by discarding the initial first second due to the random sampling for the MB-trajectories, the average c_D value is decreased compared to the MF equivalent leading to averagely higher rewards. The actions depicted in fig. 5.10 still show a periodic behavior for a MB-training. Increasing the number of models in the ensemble is able to mitigate the issue of low-pass filtering, for the training with $N_{models} = 5$ the actions show a higher degree of discontinuities as well as the ability of covering the complete range of $\omega = -5...5$. This indicates, that using a model-ensemble is able to introduce discontinuities leading to a more realistic representation of the CFD environment.

An increase of N_{models} to values greater than five leads to a deterioration of the rewards, as already mentioned. Further, the stability of the training decreased significantly, leading to a high number of crashed trainings caused by a diverging CFD simulation and consequently absence of training data for more than three CFD episodes. The log-files of the training showed, that with increasing number of models, the KL-divergence stopping criteria of the policy training routine was reached more frequently. The KL-divergence stopping criteria was introduced to bound the gradient of the policy in order to prevent too large policy updates possibly leading to instability as presented in section 2.2.1. An increase of the KL-divergence over the predefined limit leads consequently to an abortion of the policy optimization.

Introducing a model-ensemble is known to reduce the aleatoric uncertainties [21]. This means that one model may predict the next state too optimistic while the prediction of another model is too pessimistic; combining these two models yields statistically a higher prediction accuracy than using only one model. If the prediction accuracy is not sufficient, then the range of possible predicted states increases. Since each new time step, a new model is randomly taken out of the ensemble for making a prediction, the divergence within the predictions is high, leading to strong discontinuities between two consecutive time steps. Figure 5.11 shows the divergence of the predictions within the model ensemble when given the exact same input state

and corresponding action. It can be seen clearly, that although each model in the ensemble was trained on the same data, as a consequence of error propagation and uncertainties the trajectory generated by each model differs. If the model changes now every time step, instead of generating five different trajectories for $N_{models} = 5$, one trajectory is generated in total. This trajectory covers the range of all possible predictions made by each model, leading to a statistically improved prediction accuracy as explained. However, with increasing number of models in the ensemble, the probability of predicting an unrealistic value by one of the models over the course of the trajectory increases as well. The amount of these outliers is likely to increase with increasing number of models as a consequence of the diverging predictions for different models. This indicates that an increase of N_{models}, although reducing the aleatoric uncertainties, may cause stability issues to diverging predictions for similar states and actions.

Another issue which can be seen in fig. 5.11 for the trajectory generated by model number 2, is that the model is not able to generate a valid trajectory at all, probably caused by a poor prediction accuracy. If even only one state is predicted falsely by an environment model, this may lead to nonphysical or even *nan* values causing the PPO-training to crash completely. To prevent this occurrence and to improve the stability of the training it was therefore decided to check the model-generated trajectories for invalid values. Although the boundaries were chosen carefully to not influence the training process itself, filtering the trajectories for unrealistic values may introduce a bias. Another disadvantage is the dependency of such boundaries on the chosen setup, e.g. boundaries for c_D values may be depending on the Reynolds number and consequently require knowledge of the expected boundaries beforehand. This disadvantage, however, is overcompensated by potential improvement of the stability when introducing such a sanity check and is therefore accepted. The upper boundaries where derived by the maximum values encountered in the uncontrolled flow, further a safety margin was added. The lower bounds where derived using the minimum values of the best MF-case with $b = 10, l = 8s$ in episode 200 plus an additional safety margin accounting for uncertainties when determining these minimum values. This leads to the following boundaries, which the model-generated trajectories are not allowed to exceed:

- $|c_L| < 1.3$
- $2.85 < c_D < 3.5$
- $|\omega| \leq 5.0$
- $\alpha < 5000$
- $\beta < 5000$

Additionally, the trajectories where checked and discarded if they contain any *nan* values. It was further found that it is not necessary to check the states for invalid values, because if all the parameter presented here where in the defined boundaries, the states showed valid values as well.

5.1.5 Optimization of the training routine

The extension to a ME increased the performance and stability of the training routine significantly. In a final step, the training routine is modified aiming to decrease the runtime while at the same time improving the stability further. Therefore, the model architecture as well as the hyperparameters where optimized and can be found in the appendix, section A.3. The runtime was decreased by initializing the environment models with the ones from the previous CFD episode instead of training the models each new CFD episode from scratch. In episode zero, the first model in the ensemble is trained with 10 000 epochs as before, every other model in the ensemble, however, is then initialized with this first model and then trained for another 1000 epochs. In every new CFD episode, the first model in the ensemble is initialized with the first model of the previous CFD episode and then trained for 1000 epochs. The remaining models are again initialized with the first model in the ensemble and trained for 1000 epochs each. Additionally, a batch normalization and a weight decay of 10^{-3} was introduced, the batch size was chosen to be 25.

In the previous section, a sanity check of the model-generated trajectories was introduced. This lead to the unfavorable situation that for highly inaccurate models, as usually present in the first episodes due to the little amount of available training data with high variance, all trajectories were invalid and consequently discarded. The PPO-training was then forced into an infinite loop and had to be aborted manually. This problem was fixed by introducing a counter, when the environment models were not able to generate valid trajectories within 50 iterations, the PPO-training switched back to the CFD environment for the current episode. After finishing the simulation, the models were re-trained for 100 epochs each using the newly generated trajectories. This lead to an overall increase in stability of the MB-training, since the tolerated inaccuracy of the models with respect to the predictions can be controlled by the defined boundaries.

The effects of these improvements to the MB-training routine can be seen in fig. 5.12. The received rewards improved in all cases, especially for $N_{models} = 5$ and $N_{models} = 10$. These two cases are now able to outperform the MF-training significantly. Further, a monotonic increase of the rewards with increasing N_{models} can be observed as a direct consequence of an increased prediction accuracy. There are no sudden decreases in the rewards over the course of the training indicating an improved stability of the training routine. However, although yielding now higher rewards than $N_{models} = 5$, the training conducted with $N_{models} = 10$ was highly unstable in comparison the $N_{models} = 5$ as a consequence of the KL-divergence within the policy optimization discussed in the previous section.

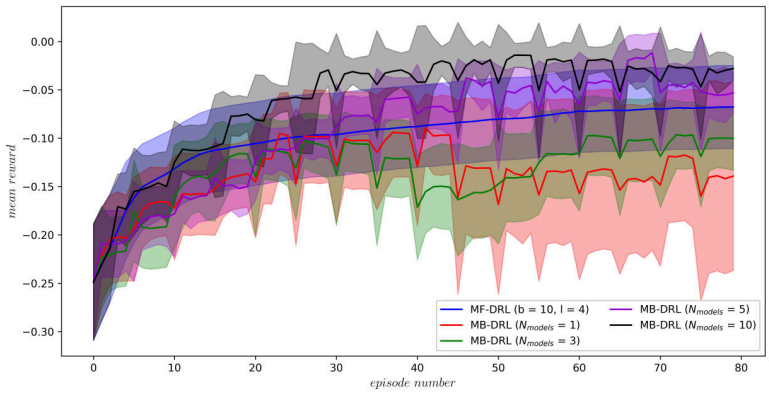


Figure 5.12: Influence of the number of models within the ensemble on the received rewards for $b = 10$ and $l = 4s$

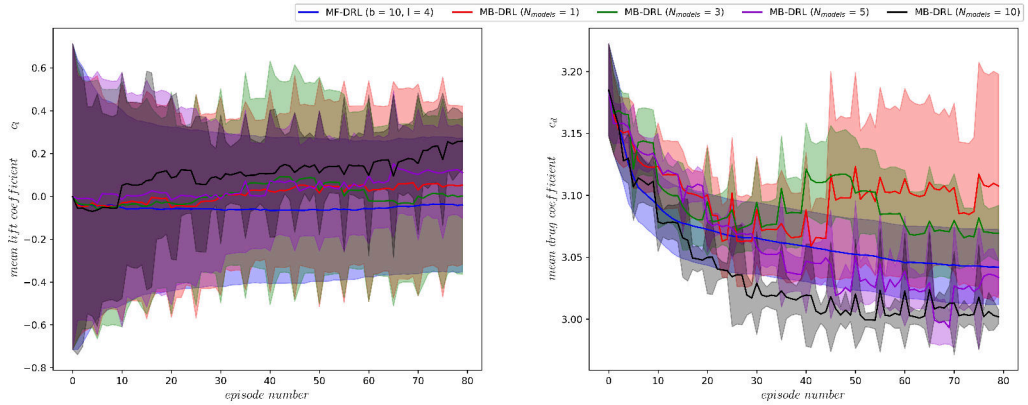


Figure 5.13: Influence of the number of models within the ensemble on c_L and c_D for $b = 10$ and $l = 4$

The c_L and c_D values with respect to the episode number are depicted in fig. 5.13. It can be seen, that with increasing N_{models} the ability to control c_L deteriorates. Especially for $N_{models} = 10$, the course of c_L is significantly diverging from values of $|c_L| \approx 0$ as achieved in the MF-training. This behavior may be an indicator that the introduction of discontinuities using a ME has a negative influence on the ability of controlling c_L. The c_D values on the other hand, monotonically decrease with increasing N_{models}, which may be another reason why $N_{models} = 5$ yields the highest training stability. If with increasing N_{models} the policy is not able to control c_L sufficiently accurate, while at the same time the performance with respect to c_D increases; it is likely that this results in an optimal N_{models} of five since c_D is weighted more than c_L.

5.1.6 Conducting a MB-training on different systems

As a consequence of diverged CFD simulations during an MB-training, a not neglectable amount of trainings crashed at some point. These errors, however, where not reproducible when running a training with the same setup and seed values on a local machine, making it hardly possible to find reasons for the unstable behavior of the MB-training routine. In the following, the problem of encountering different results when running a training on different systems will be presented briefly, these cases are all averaged over three different seed values. The seed values for trainings on the HPC cluster were the same as for the trainings on a local machine, the remaining setup was chosen to $b = 8, l = 2s$ and $N_{models} = 5$.

Table 5.2 shows the runtime for a MF-training in comparison to the MB-training. It can be seen that for both the HPC cluster and the local machine, the MB-training required significantly less runtime and resources than the MF-counterpart. When conducted on the HPC cluster, the model-based training was able to reduce the runtime by 64.16% compared to the model-free training. This discrepancy is increasing when the available computational resources are a limiting factor as it is generally the case on a local machine. The MB-training leads here to a decrease of 76.57% of the runtime compared to the MF-training emphasizing the large potential of MB-DRL for flow control applications.

The rewards of the MF-training show only minor differences when running the training on

	MF-DRL (HPC)	MF-DRL (local)	MB-DRL (HPC)	MB-DRL (local)
runtime	$04h:53min$	$23h:19min$	$01h:45min$	$05h:15min$

Table 5.2: Average runtimes for a MF- and MB-training on different systems

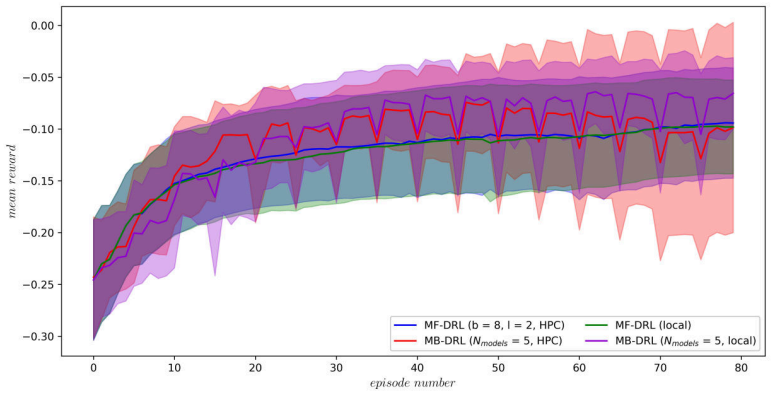


Figure 5.14: Differences between the rewards received on an HPC cluster and on a local machine

different systems, as depicted in fig. 5.14. This may be partially caused by a possible hardware-dependency of the initialization, although the chosen seed value is the same. As discussed in the PyTorch documentation [29], the initialization of the random number generator may yield different absolute values across different hardware, since the pseudo-random numbers are generated under the usage of e.g. background noise. This can cause a slightly different initialization of the policy and value network leading to diverging results. The differences within the MF-trainings, however, can be seen as neglectable.

The rewards received in the MB-training on the other hand is highly dependent on the system as can be obtained from fig 5.14 as well. This is partially caused by the fact, that in contrast to the MF-training, here the environment models need to be initialized additionally, which may also differ across different systems. Another aspect is the process scheduling with SLURM when running a training on an HPC cluster. As a consequence of an unfavorable policy caused by model inaccuracies, the CFD simulation may require significantly longer runtimes to converge compared to a MF-training. However, if there is a high traffic on the cluster, this may lead to waiting times for process execution or the necessity to interrupt processes in favor of executing other processes. After a predefined maximum execution time, which was set in all trainings to $t_{max} = 30min$, the process is canceled by SLURM. In case of an unfavorable convergence behavior of the CFD simulation, this may lead to an abortion and consequently decreased amount of available training data causing a deteriorated model accuracy and rewards. It is important to note that the average execution time of a CFD simulation is in the order of $t_{avg} \approx 3min...6min$ depending on the trajectory length, and therefore $t_{max} = 30min$ was found to be sufficiently long.

5.2 Generalization of the training routine

The stability issues of the MB-training routine as well as the poor prediction accuracy of the environment models motivated the implementation of a new training routine. The previous training routine generated the feature-label pairs directly during the model training, also the training data was statistically seen only once by the models during training. This procedure stands in direct contrast to the common practice, where all the training data is utilized each epoch in order to train the environment model and was a result of unawareness of this practice. This issue was corrected in the new training routine, further it was made use of built-in functionalities provided by *PyTorch*, e.g. the *dataloader*, rather that implementing an own customized training routine. In the new training routine, the feature-label pairs were created beforehand, leading to an additional decrease in runtime. These improvements of the training routine yield the great advantage that the model training can be conducted via a standardized interface, making it more flexible for implementing further functionalities and at the same time more robust against potential implementation errors.

Despite these adjustments, a layer normalization was used instead of batch normalization, because batch normalization is sensitive to the chosen batch size [29], which may be a disadvantage. However, the general model architecture remains the same as in previous training routine. Finally, the scheduler *ReduceLROnPlateau* was introduced in order to optimize the training routine further. The scheduler controls the learning rate based on the current validation loss. The learning rate is set to $lr = 0.01$ at the beginning of the model training. Whenever a plateau in the validation loss is reached and maintained for 10 consecutive episodes, the scheduler reduces the learning rate by a factor of 0.1. The minimal possible learning rate (lower bound) is set to $lr_{min} = 1 * 10^{-4}$.

5.2.1 Early stopping of the model-training

In the previous training routine, an early stopping criteria based on the absolute value of the validation loss was introduced, however, this criteria was never reached as a consequence of high validation losses as depicted in fig. 5.15. Both the training and validation loss of the previous training routine showed a highly discontinuous course resulting from the fact that the training data was only presented to the models once throughout the training. This discontinuous course makes it challenging to implement metrics to detect and prevent over-fitting of the environment models, since a gradient-based measurement is highly affected by the starting and end point of the interval taken for computing the gradient.

The losses of the new training routine on the other hand yield a significantly smoother course, additionally, a clear convergence behavior can be seen from fig. 5.16 within the first $\approx$ 150

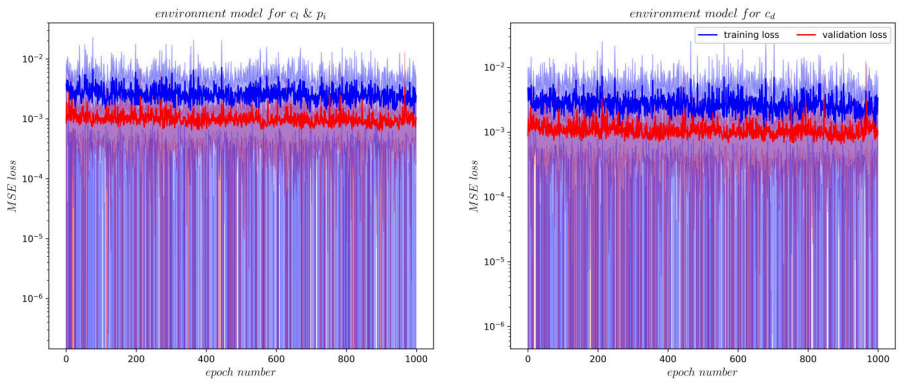


Figure 5.15: Training and validation losses using the original training routine

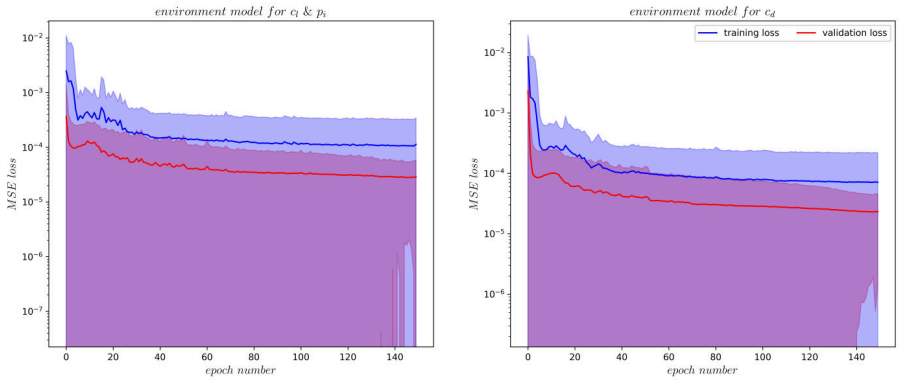


Figure 5.16: Training and validation losses using the new training routine

epochs. This improved convergence behavior makes it redundant to train each model for 1000 epochs, since it is highly unlikely that the losses decrease any further. The steady course of the losses provide the possibility to implement an additional, gradient-based early stopping criteria based on the gradient of the validation loss. In the present training routine, this stopping criteria was set to a decrease of the validation loss by less than $-1 * 10^{-6}$, averaged over the last 50 epochs. This prevents the models from over-fitting, because if the validation loss increases over the defined threshold of $-1 * 10^{-6}$, the training is aborted. As a consequence of the improved convergence behavior of the losses when using the new training routine, instead of training the first model in the ensemble for 10 000 epochs, N_{epochs} was reduced to a maximum of 2500 epochs. However, in all the following trainings, it could have been observed that the stopping criteria was generally reached after a maximum of ≈ 500 epochs. Every remaining model in the ensemble is further only trained for 150 epochs in contrast to 1000 epochs as before, since fig. 5.16 indicated a convergence within the first 150 epochs. This improvement of the model-training lead to a significant decrease in the required runtimes.

The losses of the new training routine are generally one order of magnitude smaller than the losses of the previous training routine, which indicates that the prediction accuracy of the models improved as well. In a next step, the prediction accuracy of both training routines where compared.

5.2.2 Comparison of the prediction accuracy

The PPO-training routine was simulated in order to compare the prediction accuracy of the two training routines. This means trajectories of a MF-training, which was conducted beforehand, where taken in order to provide states and actions, and then used to train the environment models successively each episode $e\%5 = 0$ until the complete 80 episodes were performed. This procedure was necessary since the environment models are always initialized with the environment models of the previous CFD episode. If the prediction accuracy of e.g. episode 80 is to be evaluated, then the models can not be simply trained with the CFD data of episode 75, because in the *real* model-based PPO-training routine, the models would have been initialized with the environment models of episode 70 and so on. For comparing the different training routines, the actions are provided by the MF-training and the same for both trainings, also the trajectories were initialized with the first 30 consecutive time steps in order to be comparable to the MF-trajectory. As already discussed in section 4.2.1, without uncertainties and error propagation the MF- and both MB-trajectories would be identical, therefore the prediction error can directly be assessed by the differences within the trajectories.

The new training routine is in the following denoted as *new training*, while the old training routine is denoted as *original training*. Figure 5.17 shows the predicted trajectory for $b =$

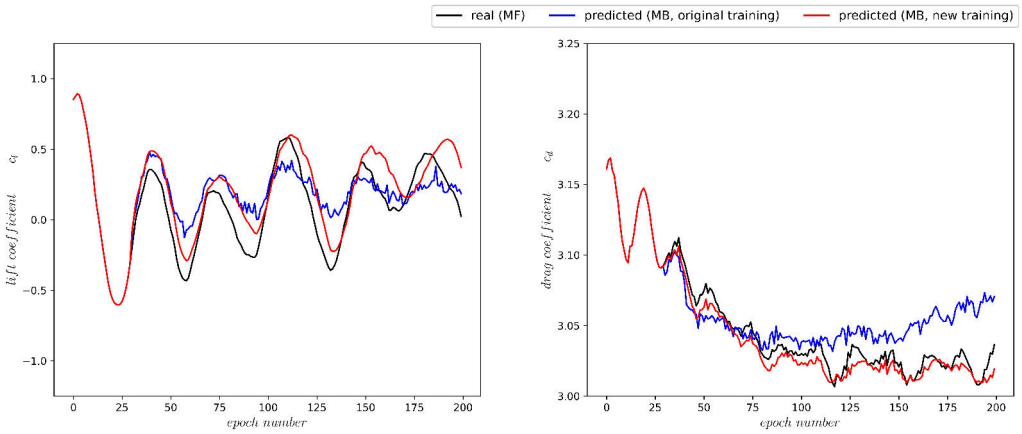


Figure 5.17: Comparison of the predicted trajectories with the original (CFD) one, for both training routines, here: $b = 8$, $l = 2s$, $N_{models} = 5$, episode 80

$10, l = 2s$ in episode 80 along with the real trajectory generated in the CFD environment. It can be seen that the prediction accuracy of the new training routine is significantly better than for the original training routine, which was already indicated by the decreased validation loss. The trajectory for c_L shows a more continuous course when using the new training routine, however, there exists a phase-shifting which is not present when using the original routine. The trajectory of c_D can be predicted accurately by the new routine, while when using the original one, the predicted trajectory starts to diverge from the real one at approximately episode 100.

For a trajectory length of $l = 6s$, however, the predictions for c_L when using the new training routine is deteriorated compared to the original one. The amplitude of c_L is predicted significantly higher than the actual amplitude, starting at approximately episode 225. Up to this point, the predictions for c_L of both training routines are fairly accurate, the original routine, again, showed a higher degree of discontinuities which are not present when using the new training routine. The prediction for c_D on the other hand is sufficiently accurate for approximately 350 time steps before diverging from the real trajectory, while the original training routine is only capable of predicting the first 200 before diverging. However, within these 200 time steps the prediction accuracy is not at a satisfying level as well, illustrating the improvement with respect to the prediction accuracy of the new training routine.

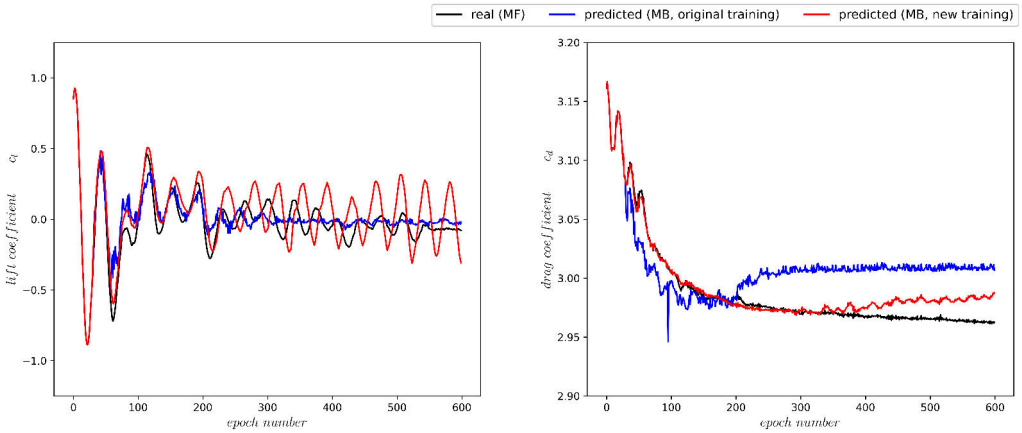
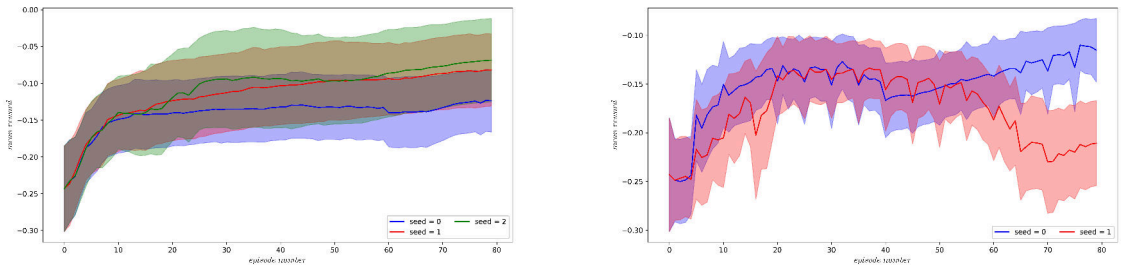


Figure 5.18: Comparison of the predicted trajectories with the original (CFD) one, for both training routines, here: $b = 10$, $l = 6s$, $N_{models} = 5$, episode 80

5.2.3 Influence of the number of models within the ensemble

Since the prediction accuracy when using the new training routine improved, it was investigated in next step if a variation of the number of models in the ensemble yields different trends than observed when using the original training routine. The dependency of the rewards on the initialization was analyzed in a first step.

The rewards for conducting a MF-training with different seed values are depicted in fig. 5.19a. Although the course of the rewards show a dependency on the seed value, the resulting differences within the trainings are generally following the same monotonic trend. This behavior changes significantly when conducting a MB-training as can be seen from fig. 5.19b. In this case, the rewards for a seed value of zero shows an overall improvement throughout the training, while the rewards of the training initialized with $seed = 1$ decrease significantly towards the end of the training. This divergence of the rewards increases with increasing number of models in the ensemble and is probably a consequence of the fact that each model within the ensemble is depending on the initialization. This results in a higher statistical variance if more models are utilized, making it challenging to derive overall trends and consequently improvements with respect to the stability of the training. It is important to note that this behavior was also observed for the original training routine, as discussed in section 5.1.6. This indicates that the prediction accuracy of the environment models does not seem to have a measurable impact on this issue.



(a) Rewards of a MF-training for different seeds **(b)** Rewards of a MB-training for different seeds

Figure 5.19: Comparison of the rewards received throughout the training for different seeds (all $b = 10, l = 2$), for the MB-training $N_{models} = 3$ was used

It could have been observed during a MB-training that the stability when using $N_{models} = 1$ is significantly worse than with $N_{models} = 5$. This trend is also displayed by the rewards shown in fig. 5.20, since the overall course of the rewards for $N_{models} = 1$ shows a significantly worse convergence behavior in terms of monotonic increase than the rewards received with a MB-training using $N_{models} = 5$ or $N_{models} = 10$. Further, the difference in the rewards between the MF- and MB-episodes is significantly higher when using $N_{models} = 1$, than e.g. for a training with $N_{models} = 5$. Although the sampling of the initial states may have an influence on the differences within the rewards for the MB-episodes, the relative difference between the training with $N_{models} = 1$ and $N_{models} = 5$ indicates, that the environment models for a training with $N_{models} = 1$ predict the course of c_L and c_D too optimistic. However, the overall trend with respect to the course of the rewards when using the new training routine is the same as already observed in the original routine, namely an increase in rewards when increasing the N_{models} to five, followed by a decrease in the rewards when increasing the N_{models} further to e.g. ten. This may be again caused by a more frequent exceeding of the threshold for the KL-divergence within the policy optimization with increasing number of models, as could have been observed from the log-files.

Figure 5.21 shows the divergence within the predictions accuracy for the different models in the ensemble, analogously to the study conducted in section 5.1.4. Generally, the predictions of

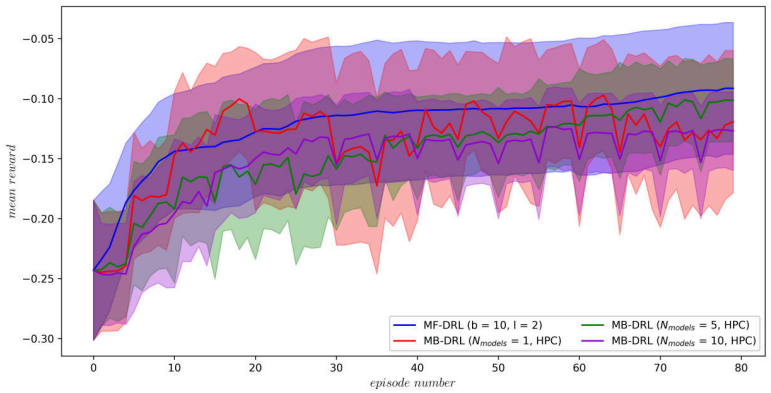


Figure 5.20: Influence of the number of models within the ensemble on the received rewards, here for $b = 10, l = 2$

the environment models are too optimistic, as indicated by the lower c_D values of the predicted trajectories compared to the real ones, causing a further increase in the rewards for MB-episodes. The divergence within the prediction accuracy is likely to be the consequence of triggering of the KL-divergence criteria when optimizing the policy, as explained in section 5.1.4. However, in contrast to the original training routine, the divergence within the predicted trajectories for the different models decreased significantly when using the new training routine, starting at approximately epoch 275. Further, all trajectories contain physical values comparable to the real trajectory. This behavior is in accordance with the observation that less trajectories where discarded as a consequence of invalid values during a MB-training when using the new training routine, making a switching back to CFD a significantly less frequent event.

An increase of the trajectory length, however, leads to a significant deterioration with respect to the training stability. As presented in the previous section, the predicted trajectory starts to diverge from the real one at approximately epoch 300, when using $N_{models} = 5$ and the new training routine. The trajectories with $l = 4s$ on the other hand contain 400 values, making it necessary to be able to predict 370 consecutive time steps sufficiently accurate. Fig. 5.22 shows the rewards received for a MB-training with $N_{models} = 1$ and $N_{models} = 5$. It is important to note that the results presented for $N_{models} = 1$ could not have been averaged over multiple seed values, since it was not possible to achieve another stable training despite the one depicted in fig. 5.22. At some point during the training, all trainings crashed due to a divergence of the CFD simulations over three subsequent CFD episodes. Consequently, resulting from the

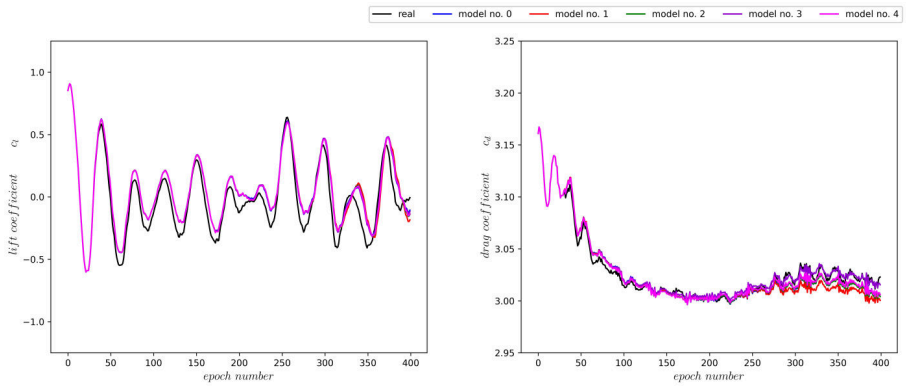


Figure 5.21: Prediction accuracy of the different models when using the new training routine for episode 80, $l = 4s$ and $b = 10$

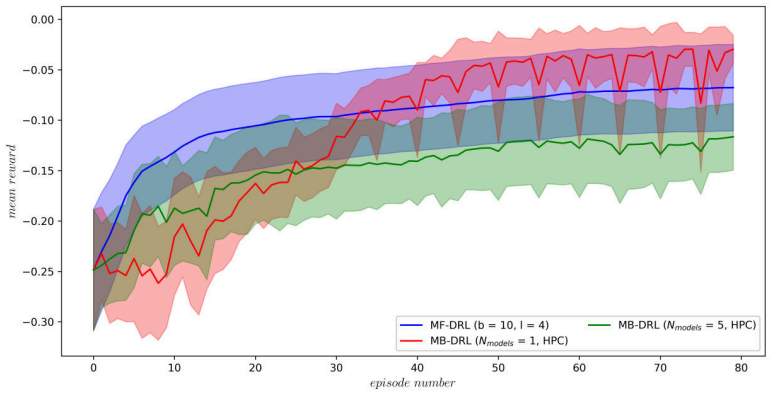


Figure 5.22: Influence of the number of models within the ensemble on the received rewards using $b = 10$ and $l = 4$

high dependency of the results on the initialization, the results for this case can not be seen as trustworthy enough to draw any conclusions from it. When using $N_{models} = 5$ for the training, the stability improved significantly in comparison to $N_{models} = 1$, the results presented in fig. 5.22 for $N_{models} = 5$ were therefore averaged over three different seed values. Although the rewards for a training with $N_{models} = 5$ are not reaching the level achieved in a MF-training, the overall stability when using the new training routine improved. $N_{models} = 10$ showed again a significant deterioration with respect to the stability throughout the training, leading to the unfavorable situation that it was not possible to obtain even one full training and is therefore not considered here.

A network architecture study has additionally been carried out aiming to improve the stability of the training further. However, the results did not yield any trends, comparable to the behavior observed in section 4.2.2. The results of this network architecture study are presented in the appendix, section A.6.1 for the sake of completeness. It is important to note that these results may be depended on the trajectory length and buffer size, making it a challenging task to find a network architecture which performs well over a vast variety of different settings. In a next step, the last remaining parameter, namely the alternation between MF- and MB-episodes, which may be able to influence the training stability was investigated.

5.2.4 Alternation between model-based and model-free episodes

The alternation between the model-free and model-based episodes was seen as another possible reason for the stability issues of the MB-training. Resulting from the increasing divergence between the policy under which the training data was collected and the policy used when generating the MB-trajectories, the accuracy of the environment models decrease with increasing distance to the last CFD episode. This may cause stability problems due to an exploitation of the model-uncertainties by the policy.

The rewards with respect to the episode number are depicted in fig. 5.23, the ratios of MB/MF episodes correspond hereby to switching to the CFD environment every episode $e\%2 = 0$, $e\%5 = 0$, $e\%6 = 0$ and $e\%8 = 0$ followed by an update of the environment models with the trajectories of the current and last CFD episode. In all cases, the first episode is conducted in CFD as well in order to initially train the environment models. The training with a ratio of $MB/MF = 6.27$, corresponding to conducting all episodes $e\%8 = 0$ in the CFD environment, was averaged over two different seed values as a consequence of severe stability issues of the training. This indicates that with increasing distance of two subsequent CFD episodes, the environment models become too inaccurate and therefore vulnerable to an exploitation by the policy. The rewards,

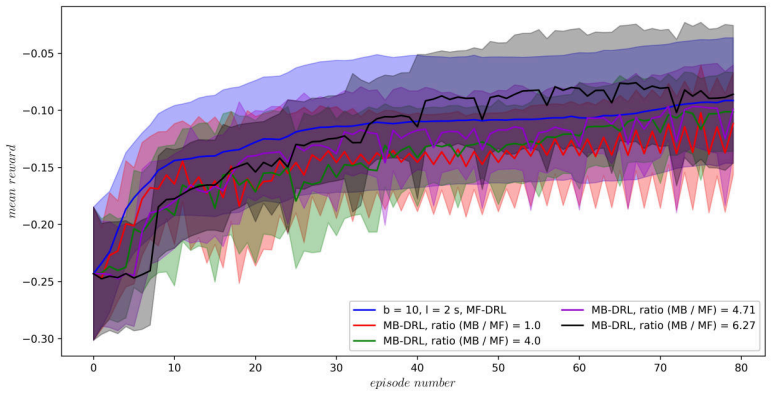


Figure 5.23: Influence of the ratio between MB- and MF-episodes for MB-trainings with $b = 10$ and $l = 2$

however, increase with increasing ratio of MB/MF episodes, for $MB/MF = 6.27$, the MB-training reaches even a comparable performance to the MF-counterpart. It was not able to determine the reason for this behavior, but the standard deviation for the training with the ratio $MB/MF = 6.27$ is significantly larger than for all other trainings. Since this training has a relatively high amount of MB-episodes it may be increasingly dependent on the initialization in contrast to cases with lower MB/MF ratios. Therefore, the high rewards achieved may be a consequence of a favorable initialization considering that this case was only average over two instead of three different seed values.

Figure 5.24 depicts the lift and drag coefficients corresponding to the trainings shown in fig. 5.23. It can be seen that except for the ratio of 6.27, all MB-trainings are able to reduce and maintain the course of c_L at a level in the vicinity of $|c_L| \approx 0$. The reason for the significantly different behavior of the training with the ratio of 6.27 could not been found, both seed values for this training showed a worse ability to control c_L. The results for c_D are generally following the trend already displayed by the rewards. As discussed, the training with a ratio of 6.27 shows a significantly larger standard deviation, which is probably caused by the higher dependency on the initialization due to a larger amount of MB-episodes.

The stability of the training with a trajectory length of $l = 2s$ showed minor improvements with decreasing ratio of MB/MF episodes. At the same time, the rewards decreased while

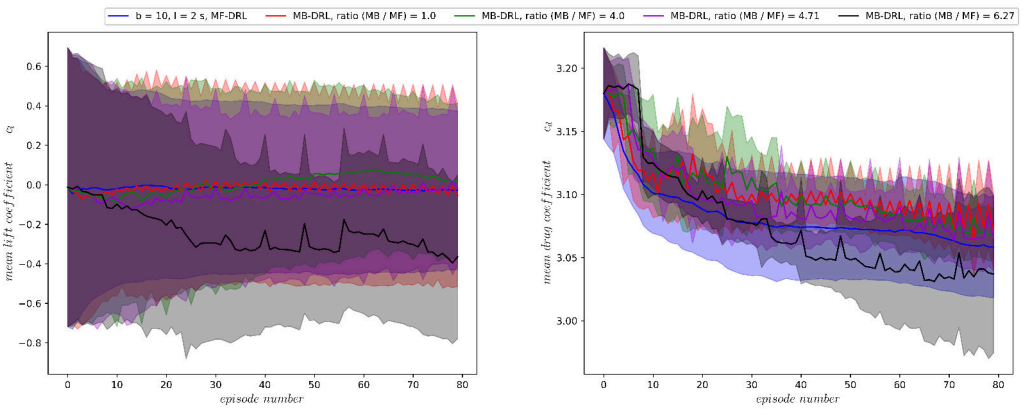


Figure 5.24: Lift and drag coefficients with respect to the episode number for MB-trainings with $b = 10$ and $l = 2s$ and different ratios of MB/MF episodes

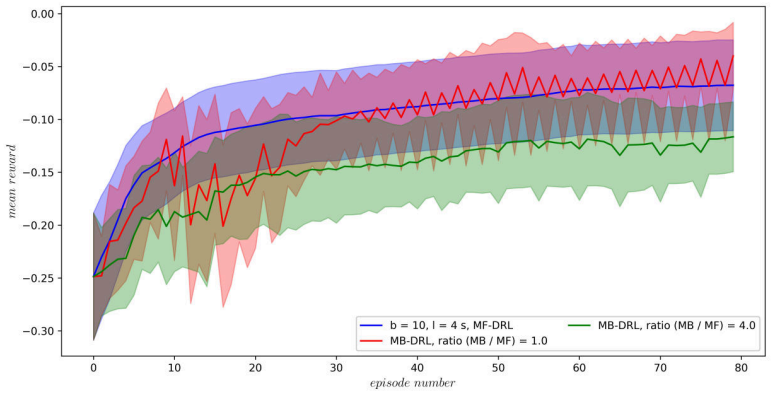


Figure 5.25: Influence of the ratio between MB- and MF-episodes for MB-trainings with $b = 10$ and $l = 4s$

the runtime increases with a more frequent switching between MB- and MF-episodes resulting in the aforementioned optimization problem. When increasing the trajectory length to $l = 4s$, however, this trend reverses as depicted in fig. 5.25. The stability for trainings with $l = 4s$ was significantly worse than with $l = 2s$, leading to the unfavorable situation that there are only data for $MB/MF = 1$ and $MB/MF = 4$ available. All trainings with other ratios crashed at some point in the training, further the results for $MB/MF = 1$ are only averaged over two different seed as a consequence of these stability issues. The training with a ratio $MB/MF = 4$, corresponding to all episodes $e\%5 = 0$ conducted in the CFD environment, did not encounter any stability issues and was therefore averaged over three different seed values. Although the amount of data is not sufficient to draw a profound conclusion, it is indicated that for $l = 4s$, with increasing ratio of MB/MF episodes the rewards decrease while the stability of the training increases. This behavior stands in direct contrast to the trends observed for trainings with $l = 2s$.

Since the ratio $MB/MF = 4$ was able to run fairly stable independent of the trajectory length while at the same time yielding acceptable rewards, it was decided to maintain this setup. This means in the following, all episodes $e\%5 = 0$ are conducted in the CFD environment while for the other episodes the trajectories were generated by the environment models. In order to ensure that the instabilities where not depending on the HPC cluster, in a final step the training routine was migrated to AWS, as presented in the next section.

5.2.5 Improvements of the training routine and migrating to AWS

A dependency of the training performance on the system (local machine versus HPC cluster) was presented in section 5.1.6. It could have been observed in general that in some cases the training hung up in episode zero for no obscure reason when conducting a MB-training on the *Phoenix* HPC cluster of TU Braunschweig. This behavior motivated the necessity to compare the MB-trainings run on the HPC of TU Braunschweig to trainings conducted on *AWS* in order to ensure that the encountered stability issues were not a consequence of the available computational resources. The training routine was therefore migrated to *AWS*, the setup remains the same as for trainings conducted on the *Phoenix* cluster; the only difference is the usage of 10 CPU's for model-training instead of only 4 CPU's.

The results of MB-trainings conducted on *Phoenix* in comparison to *AWS* are depicted in fig. 5.26. The overall course of the rewards until episode 40 displays significant differences for the different systems, however, these difference converge for episodes > 40 leading to similar results towards the end of the PPO-training. It is important to note that for both HPC systems, all data was averaged over the same three different seed values. One possible explanation of what caused the differences within the rewards may be the fact that during training, some CFD simulations diverged and consequently did not yield any results. This leads to a reduction of available data for model training and sampling of the initial states, and therefore to a decrease in model-accuracy resulting in deterioration of the rewards as already discussed in section 5.1.6. However, the absolute amount of diverged CFD simulations is not the only factor influencing the performance and stability of the MB-training, it is also necessary to account for the episodes in which the simulations diverged.

At the beginning of the PPO-training, the variance within the data is high, leading to generally more inaccurate environment models than for episodes towards the end of the PPO-training. If a simulation diverges in an early stage of the PPO-training it is likely that this deteriorates the model-accuracy significantly, since the trajectories are highly heterogeneous. In later episodes, the trajectories show a qualitatively similar behavior making a lack of available data not as severe as in the first episodes. Divergence of CFD simulations within the first CFD episodes may has a significant impact on the overall stability of the training as a consequence of error propagation and the fact that the amount of episodes which still need to be conducted is large at this point in the training process; influencing the course of the rewards and stability. On the other hand, if a CFD simulation diverged towards the end of the PPO-training, e.g. in episode

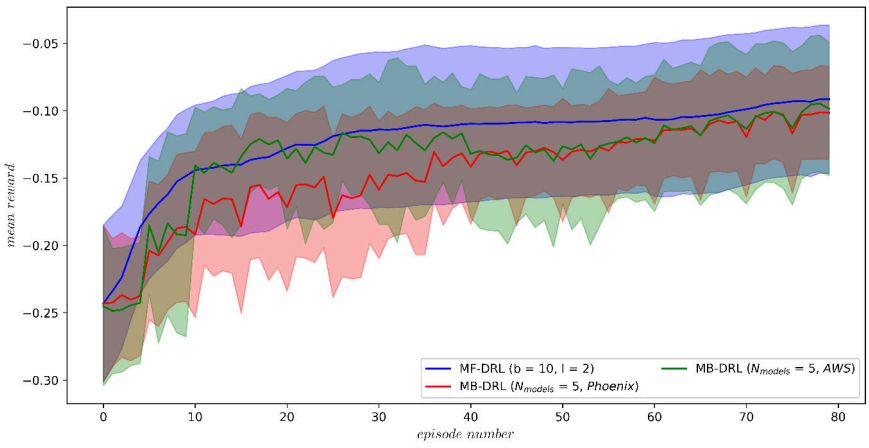


Figure 5.26: Rewards received for MB-trainings on the *Phoenix* cluster in comparison to *AWS* using $b = 10$ and $l = 2s$

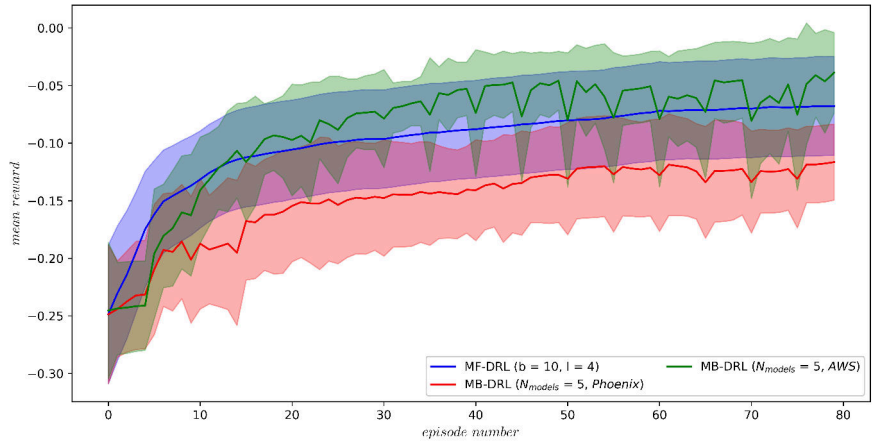


Figure 5.27: Rewards received for MB-trainings on the *Phoenix* cluster in comparison to *AWS* using $b = 10$ and $l = 4s$

75, the impact on the remaining PPO-training can be seen as rather low since the environment models are significantly more accurate caused by a low variance within the data. Further, the impact of the overall course of the training is not given, since the training at this point is almost finished. Another aspect is the point within the trajectory, namely the amount of time steps, after which the simulation diverges. If the simulation diverges e.g towards the end of the trajectory, there is still a reasonable amount of data available, in contrast to the situation when the simulation divergence after the first few time steps. Consequently, although the stability of the training routine is the same across different systems in terms of diverged CFD simulations, the results may differ significantly if the simulations diverged in different episodes containing a different amount of time steps. However, when conducting MB-trainings on *AWS*, the issue that the training hung up in episode zero for no obscure reason could not been observed, indicating that the training stability may be influenced by the HPC cluster where it is conducted.

Trainings with a trajectory length of $l = 4s$ showed a significant divergence between rewards received on *Phoenix* and on *AWS* as depicted in fig. 5.27. This is mainly caused by stability issues for different seed values on those two different systems, leading to the situations that seed values converged on*Phoenix* did crash on *AWS* and vice versa. In order to have the same amount of data for all the here shown trainings, all cases were averaged over three seeds, but the seed values itself were not necessarily the same as discussed at the beginning of section 5.1.2 as a direct consequence of the convergence behavior when conducting a training on different systems. The significant discrepancy between rewards received in a training on *AWS* and a training conducted on *Phoenix* is mainly reasoned by the usage of different seed values. However, this emphasizes that results may be depending on the system and are therefore not entirely reproducible. Nonetheless, fig. 5.27 shows that the new training routine for the MB-training is able to reach a similar performance as the MF-counterpart even with a trajectory length of $l = 4s$.

Since the stability of the training routine is still a major concern, despite the HPC cluster on which the training is conducted, three approaches have been developed in a final step aiming to solve or at least mitigate the encountered stability issues. These ideas, however, did not yield the expected results but this may be changed by a more thorough investigation of these approaches in the future and are therefore presented briefly in the following.

Increase the weight of ω and c_D within the c_D−environment models

The environment models take the states, c_L, c_D and ω of 30 subsequent time steps as input. However, the states consist of 12 values for each time step while c_L, c_D and ω are only represented by one value for each time step. The course of the trajectories for c_D and ω are significantly more complex than the trajectories of the states and c_L. These two facts raised the concern that the environment model for predicting c_D is not able to learn the dependencies between c_D and ω sufficiently accurate since the amount of the input values are rather small compared to the amount of input values for the states an c_L. This motivated to split the fully connected model internally into two networks. The model input and output remains the same but internally, the model is split up into one model for c_L and the states, and another model for c_D and ω. Each sub-model consists of multiple layers with different number of neurons. The resulting output of these two sub-models is then merged into one vector, which is finally used as input into the second part of the model. This second part outputs then the predicted c_D value. In the following, this model is denoted as *new model*. The sub-model for processing the c_L values and states contains two hidden layers with 50 neurons each while the sub-model for handling c_D and ω consists of 3 hidden layers with 50 neurons each. The output of those sub-models is merged and used as an input into the second part of the model, which has two hidden layers with 100 neurons each. These changes of the model architecture only apply the to the environment models for predicting c_D, the model for predicting c_L and the states remained the same fully connected network used in the previous sections.

This new model for predicting c_D offers a higher flexibility and possibilities to optimize the model architecture further. Although this approach yielded promising results as discussed in the following, due to time constraints it was not able to investigate the influence of the model architecture on the rewards and training stability more thoroughly.

Placing additional probes in the vicinity of the cylinder

Up to this point, 12 probes were placed in the wake of the cylinder, these pressure values are used as input into the policy network as well as for the environment models. Considering that the domain consists of 11076 grid cells, this may be not enough data points to be able to fully represent the current state of the simulation by only these 12 data points. In order to provide more data points on the current state of the simulation, another 12 probes were placed in the vicinity of the cylinder as depicted in fig. 5.28.

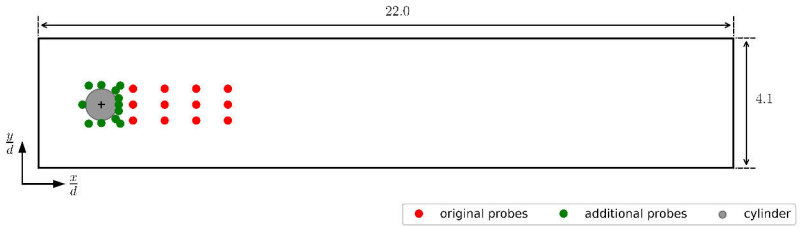


Figure 5.28: Location of the additional probes (green) placed in the vicinity of the cylinder

Correct the error between trajectories generated in CFD and by the models

Inspired by the feature extractor of the AMPO algorithm [42], additional models for correcting the model-generated trajectories were introduced. These models were trained during the PPO-training routine after each episode conducted in the CFD environment on the difference between the trajectories generated by the environment models and the real ones. The correction models for c_D and c_L consist of 5 hidden layers with 50 neurons per layer while the model for correcting the states has 5 hidden layers with 150 neurons per layer. Figure 5.29 depicts a trajectory

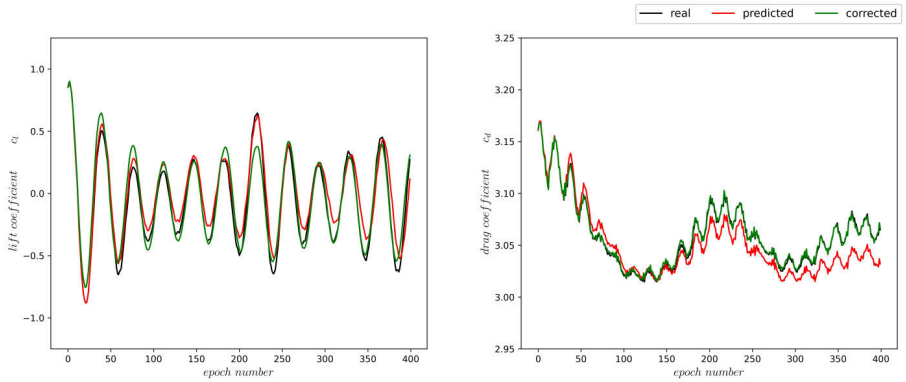


Figure 5.29: Prediction accuracy of the environment models with and without additional correction models, here with $N_{models} = 5$ and $l = 4s$ for episode 25

generated within the CFD environment, denoted as *real* along with the predicted trajectory of the environment model without additional correction models; denoted as *predicted*. Further the corrected trajectory, denoted as *corrected* is shown. Clearly, the correction models are able to correct the model-generated trajectory highly accurate, especially for the trajectory of c_D. However, these correction models can only be updated in the PPO-training routine every CFD episode. This leads to the issue that while the policy changes over the MB-episodes, the correction models are not able to adapt to this new policy, which increases the error made by the correction models over the course of the MB-episodes.

Figure 5.30 shows the rewards received when conducting a MB-training with these different approaches for $l = 4s$. The trainings with a trajectory length of $l = 2s$ yielded a qualitatively similar behavior and can be found in in the appendix, section A.6.2. Further, the results for a training with $l = 6s$, fully connected models and $N_{probes} = 12$ are depicted in the appendix, section A.6.2 as well, emphasizing the increasing influence of the seed value with increasing trajectory length. As it can be seen in fig. 5.30, the rewards are generally lower with increasing number of probes placed in the flow field. This may be caused by a redundancy within the data of the probes, meaning the additional probe yield no or only partially new information on the state. Since the number of probes doubled, this results in a doubling of the input states as well, which may cause a decreasing influence of c_D and ω relatively to the probes. The application of

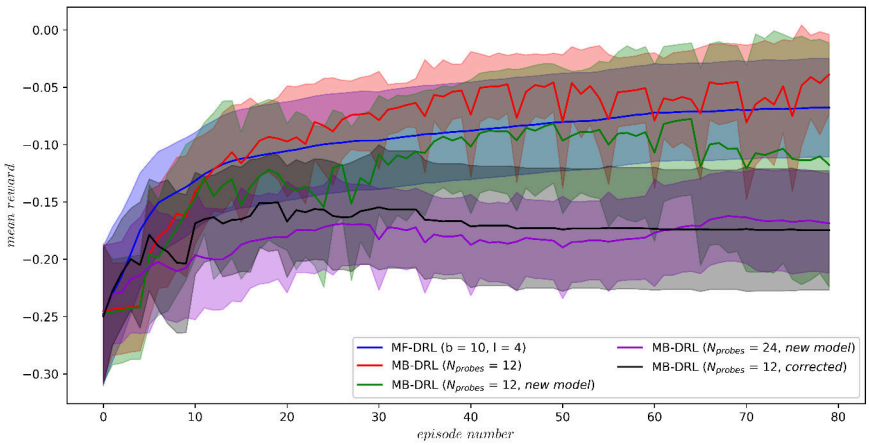


Figure 5.30: Comparison of the rewards for MB-training using the different approaches, here for a training with $b = 10$ and $l = 4$

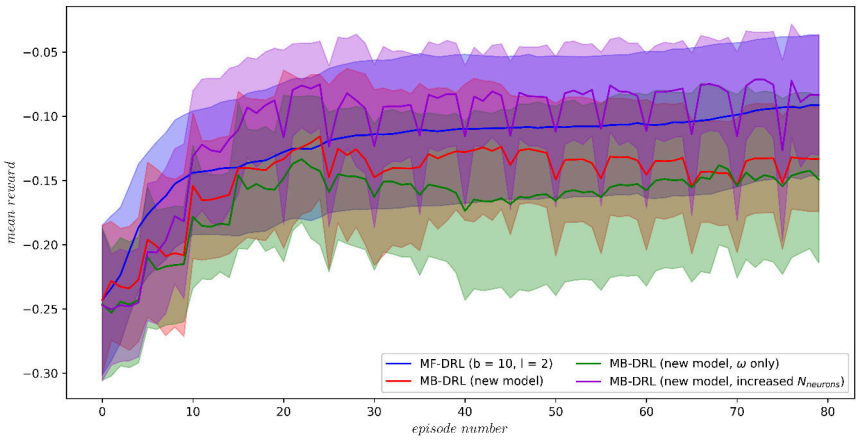


Figure 5.31: Comparison of the rewards for MB-training with differently weighted ω and c_D, here for a training with $b = 10$ and $l = 2$

additional correction models yields comparably low rewards as placing additional probes in the flow field, which may be a consequence of the increasing error made by the correction models over multiple consecutive MB-episodes, as already discussed. Increasing the weight of ω and c_D within the environment model, however, showed a high degree of stability over the course of the training although the rewards are not as high as when using a fully connected model. Since the lower rewards may be a consequence of an unfavorable network architecture, the influence of the network architecture was briefly investigated in the following.

The model architecture with respect to the number of layers and neurons remained the same throughout this study, only the number of output neurons for each sub-model was altered. The environment models for predicting c_L and the states remained a fully connected network, only the environment models for predicting c_D where changed, as already discussed. The first training depicted in fig. 5.31, denoted as *new model* is the same as already presented. In this case, the sub-model for ω and c_D contains 60 output neurons while the sub-model for c_L and the states has 390 neurons. For the second case, denoted as *new model ω only*, the first sub-model handles c_D, c_L and the states while the second sub-model only processes ω. The number of output neurons remained the same, but since c_D was moved to the first sub-model, this case corresponds to an increase of weight for ω. The last case, denoted as *new model increased* $N_{neurons}$, is generally the same as the first case, the only differences lie within the number of output neurons for the $c_D - \omega$ sub-model. Here, $N_{neurons}$ was increased to $N_{probes} * (N_{actions} + N_{c_D})$ leading to an overall $N_{neurons}$ of 720. The number of output neurons for the $c_L - p$ part of the model remained with $N_{neurons} = 390$ the same. It can be seen clearly in fig. 5.31 that an increase of the number of output neurons for the $c_D - \omega$ sub-model increased the rewards significantly compared to the other two cases. The stability is higher for trainings with a trajectory length of $l = 2s$, but when increasing the trajectory length to $l = 4s$, for which the results are provided in the appendix, section A.6.2; the stability deteriorates to a level even worse than for trainings with fully connected models.

It is important to note that this parameter study was conducted at the end of this thesis, after assessing the final policies for a Reynolds number of $Re = 100$. It was not possible to benchmark the final policies using this new model architecture as well as analyzing the influence of the model architecture more thoroughly, due to time constraints. The results up this this point, however, indicated a significant improvement with respect to the overall training stability for trajectory lengths of $l = 2s$ while at the same time reaching a comparable performance and runtimes. An extension to the model for predicting c_L and the states may improve these results further.

5.3 Comparison of the final results for Re = 100

In order to analyze the performance of the final policies, the trainings conducted on AWS with fully connected networks and 12 probes were taken. These trainings are the same as presented in the previous section and taken since they yielded comparable results to their MF-counterpart. While the rewards were averaged over three different seed values, the final policy was determined analogously to the procedure presented in section 5.1.3, meaning the best seed value of each case was taken. The seed values can be found in table 5.3, the policies for the MF-cases where taken from episode 80 while for the MB-trainings the last CFD episode, namely episode 75 was used.

Figure 5.32 summarizes the rewards received over the course of the MF- and MB-trainings. These rewards have already been discussed in the previous section and are here presented again in a more compact way for a better understanding. It can be seen that for a trajectory length of $l = 2s$ the rewards converged on a slightly lower level than in the MF-training while for $l = 4s$, the MB-training is able to reach the performance of the corresponding MF-training. Despite the deteriorated stability of the MB-training, this shows the general feasibility of MB-DRL for flow control applications.

Fig. 5.33 shows the course of the final c_L and c_D trajectories for the training with $l = 4s$ in comparison to each other and to the uncontrolled flow, the control starts hereby at $t = 4s$. The course of c_L for the MB-training is not as periodic as when conducting a MF-training and shows outliers towards higher c_L-values. This indicates that the policy for ω is not optimal yet when using the MB-training. For the c_D-trajectory, the agent trained with the MB-training yields significantly lower values than for the policy of the MF-training, however, the course of c_D shows a less periodic and more unstable behavior. The results for policies of MB- in comparison to MF-trainings using a trajectory length of $l = 2s$ can be found in the appendix, section A.7 along with the c_L- and c_D-coefficients with respect to the episode. These results are qualitatively similar, however, the course of c_D yields oscillation with a low frequency for the policy of the MB-training, indicating that the performance of the policy is deteriorated compared to the MF-counterpart. This was already indicated by the lower rewards received in the MB-training.

The mean c_L and c_D values averaged over the quasi-steady flow between $t = 6...12s$ can be found in table 5.3 for the uncontrolled and controlled flow along with the corresponding standard deviation. The MF-training with a trajectory length of $l = 2s$ was able to reduce the mean c_D by 5.72%, but the mean c_L, however, yields an increase by 660.38% compared to the uncontrolled flow. The standard deviation was reduced by 22.22% for c_D and 69.38% for c_L when applying active flow control. An increase of the trajectory length to $l = 4s$ leads to a reduction of the

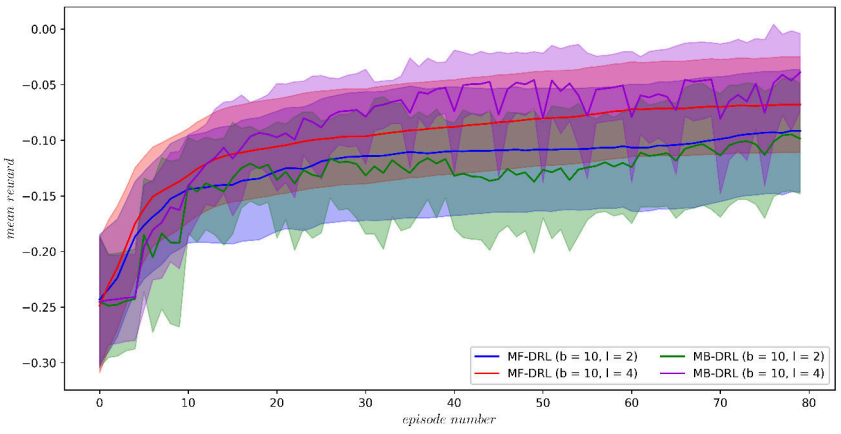


Figure 5.32: Comparison of the rewards for the final MF- and MB-trainings at $Re = 100$

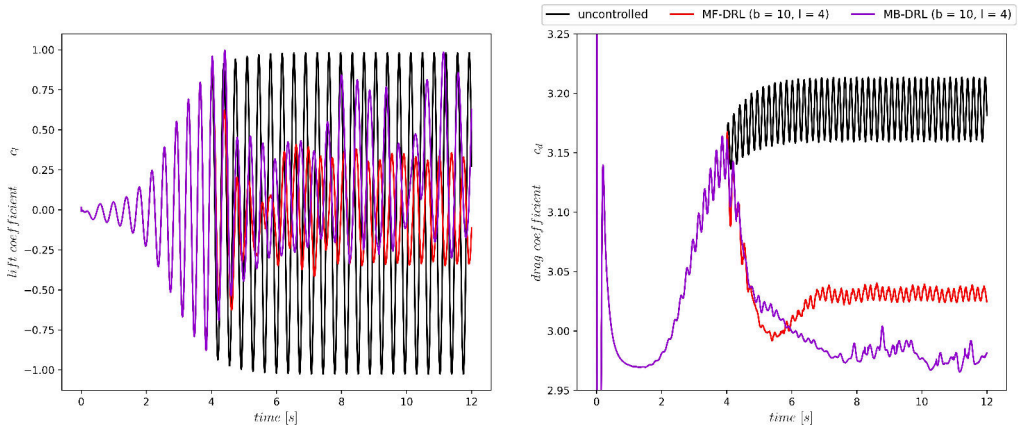


Figure 5.33: Lift and drag coefficients of the uncontrolled flow in comparison to the controlled one for a training with $l = 4$

mean c_D by 4.94% and the mean c_L by 81.13% in a MF-training while the agent was able to decrease the standard deviation by 67.74% for c_D and 66.29% for c_L. These load reductions are in accordance with the results achieved in the previous work of [6] and [9].

As already indicated by fig. 5.32, the ability to control the course of the c_L-trajectory is worse when conducting a MB-training in comparison to a MF-training. This trend can be confirmed when analyzing the average load reductions of the final policies of the MB-trainings. In comparison to the MF-training, the MB-training with $l = 2s$ was able to decrease the mean c_D further by 0.07%, however, the corresponding standard deviation increased by 85.71%. The mean c_L increased by 457.14% in comparison to the MF-training while the standard deviation for c_L increased by 139.91%. This shows that the MB-training is not able to control the control the course of c_L as efficiently as the policy of the MF-training as already mentioned. On the other hand, by using the MB-training routine, it was able to decrease the average runtime by 56.36% in comparison to the MF-training.

For the MB-training with a trajectory length of $l = 4s$, the mean c_D could have been decreased by 1.56% compared to the MF-training, while the corresponding standard deviation increased by 33.33%. The mean c_L resulting from the MB-training is with 9750% significantly increased compared to the performance when conducting a MF-training. The increase of the standard deviation of c_L is with 138.75% comparable to the training with a trajectory length of $l = 2s$. As a consequence of the increased trajectory length and consequently the computational requirements for the CFD simulation, by utilizing the MB-approach the average runtime could have been even decreased by 68.91%, emphasizing the great potential of MB-DRL with respect to the computational requirements. In the current implementation, however, the MB-trainings are significantly worse in controlling c_L than the MF-trainings, but on the other hand show an improved ability in controlling c_D.

case	seed	$\mu(c_D)$	$\mu(c_L)$	$\sigma(c_D)$	$\sigma(c_L)$	$\mu(runtime)$
uncontrolled	–	3.1863	−0.0106	0.0186	0.7119	–
MF-DRL, $l = 2s$	2	3.004	0.07	0.014	0.218	$03h:04min$
MF-DRL, $l = 4s$	1	3.029	0.002	0.006	0.24	$09h:05min$
MB-DRL, $l = 2s$	0	3.002	−0.32	0.026	0.305	$01h:20min$
MB-DRL, $l = 4s$	1	2.982	0.195	0.008	0.333	$02h:49min$

Table 5.3: Average c_L and c_D values of the final policies in the quasi-steady state at $t = 6...12s$

5.4 Model-based training at higher Reynolds numbers

In a final step of this thesis it was investigated if the MB-training can be applied to flows with a higher Reynolds number, therefore, the MF-training is bench-marked in a first step with respect to the Reynolds number. Additionally to the changes within the numerical setup presented in section 3.1.3, when increasing the Reynolds number it is not necessary to run the simulation until the same physical end time is reached. Instead it is sufficient to keep the dimensionless end time constant since the numerical time step was adjusted to ensure that the amount of interactions between the agent and environment is held constant independently of the Reynolds number. This means that e.g. a trajectory length of $l = 0.4s$ for $Re = 500$ corresponds to $l = 2s$ at $Re = 100$. The quantities used for non-dimensionalizing the time is presented in the appendix, section A.1. In order to account for the finer mesh, the number of sub-domains was further increased from two ($Re = 100$) to four ($Re = 500$).

The Reynolds number was firstly increased to $Re = 500$. Since the inflow velocity increased corresponding to the Reynolds number, the boundaries of the action had to be increased as well. The influence of ω on the received rewards in a MF-training is depicted in fig. 5.34. Clearly it can be seen that despite the training with $\omega \in [-150, 150]$, all trainings crashed at some point. The crash of the PPO-training routine is hereby caused by a divergence of the CFD simulations analogously to the behavior observed when conducting a MB-training. In contrast to the MB-training, the MF-training only utilizes trajectories of the current episode as discussed in section 2.2.2, making it not possible to re-use trajectories of the previous episode when there is no data for updating the agent available in the current episode. It is important to note that all cases presented in fig. 5.34 are only averaged over one seed value as a consequence of the poor stability of the training. It can further be seen that the episode in which the crash occurs is increasing with increasing ω. On the other hand the course of the rewards show an increasingly discontinuous behavior with increasing ω, a reason for this behavior could not have been determined. Despite the stability issues, it can be seen that that ω is required to be at least in the order of $|\omega| \geq 25$ to identify an improvement of the rewards throughout the course of the training.

The log-files of the trainings showed no errors or anomalies encountered, an additional increase of the maximum execution time did also not yield any improvements with respect to the stability of the training. In order to rule out that the trajectory length may have been a potential reason for this behavior, the influence of the trajectory length on the stability of the MF-training was

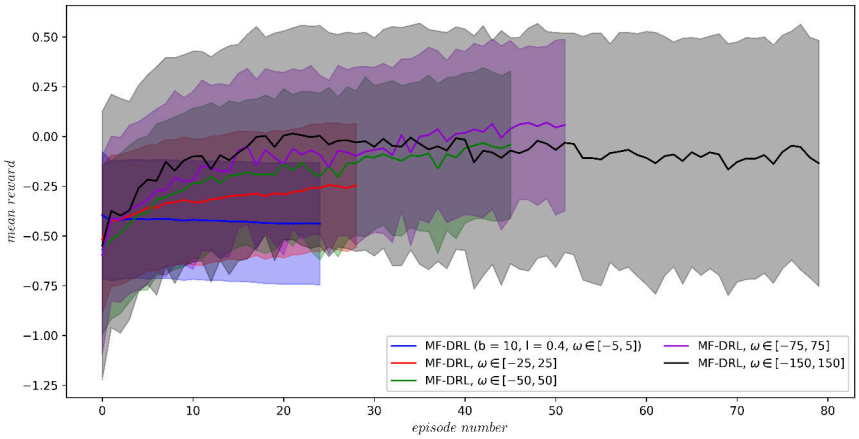


Figure 5.34: Reward received for a MF-training with a Reynolds number of $Re = 500$ with respect to the chosen ω using a trajectory length of $l = 0.4s$

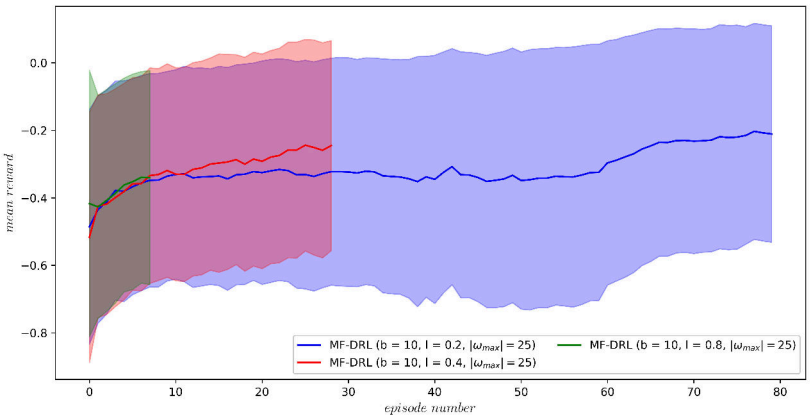


Figure 5.35: Reward received for a MF-training with a Reynolds number of $Re = 500$ with respect to the chosen trajectory length using $\omega \in [-25, 25]$

investigated. The results of this analysis are depicted in fig. 5.35. As shown, the stability of the PPO-training decreases significantly with increasing trajectory length. The trainings shown in fig. 5.35 are again only averaged over one seed values as a consequence of the poor stability of the trainings. Further, in this figure only the best seed value of each case in terms of stability is depicted, this means that all other seed values showed an even worse stability. A reason for this behavior could also not have been determined.

It was investigated in a final step if the training stability can be improved when initializing the training with a pre-trained policy instead of a random one, as suggested by [49]. Therefore, the final policy for a MF-training at $Re = 100$ using $l = 2s, b = 10$ and $\omega \in [-25, 25]$ was utilized in order to initialize the MF-training at $Re = 500$. The trajectory length of $l = 2s$ at $Re = 100$ is hereby equivalent to $l = 0.4s$ at $Re = 500$ as discussed at the beginning of this section. The results of this study are depicted in fig. 5.36, clearly it can be seen that the MF-training using a random policy crashed in episode 28, while for the MF-trainings initialized with the final policy of the MF-training at $Re = 100$ all seed values were able to run complete the trainings without encountering stability issues. This indicates that with increasing Reynolds number

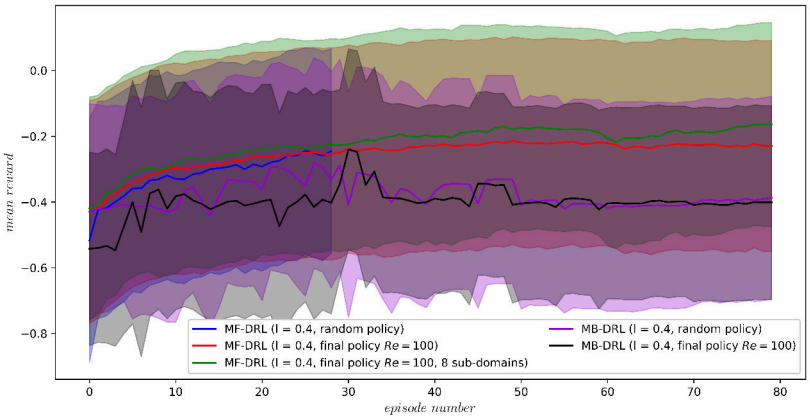


Figure 5.36: Reward received for a MF- and MB-trainings with a Reynolds number of $Re = 500$ with respect to the policy used for initialization. The actions are in all cases $\omega \in [-25, 25]$.

random actions are rather unfavorable, which may be caused by the increasingly complex system dynamics at higher Reynolds numbers. The MB-trainings, however, yielded the same stability issues and comparable rewards despite the initialization of the policy. Both MB-trainings could have been therefore only averaged over one seed value. Fig. 5.36 further shows the influence of the number of sub-domains used on the rewards. In contrast to trainings with $Re = 100$, for $Re = 500$ the numerical domain was decomposed into four sub-domains instead of two, leading to an increased inter-process communication during runtime, which may contributes to the encountered stability issues. A MF-training with eight sub-domains instead of four was therefore additionally conducted, which utilized the exact same setup and seed values in order to investigate this potential issue. As can be seen in fig. 5.36, although the rewards are slightly higher when using eight subdomains, the yield a more unstable course. Further, this training was only averaged over two seed values, since the training with $seed = 0$ crashed in episode 74, which may also be the reason for the increased rewards. Overall, this indicates a dependency of the stability of the MF-training on the number of sub-domain, since when using four sub-domains the MF-training with $seed = 0$ did not crash.

The Reynolds number was finally increased to $Re = 1000$, but the results with respect to the stability and received rewards showed a qualitatively similar behavior to the MF-trainings at $Re = 500$ and was therefore not further investigated. However, for the sake of completeness the influence of ω on the received rewards with respect to the episode can be found in the appendix, section A.22.

5.5 Comparison of the final results for Re = 500

In the following, the final policies of the training conducted in the previous section were assessed with respect to the ability of efficiently controlling the flow at $Re = 500$. Since the rewards of the MB-trainings showed no or only a neglectable improvement over the course of the training they are not considered here, because it is highly unlikely that these policies yield any helpful results. The rewards of the MF-trainings on the other hand were able to improve throughout the training and are therefore chosen. Since the training using the final policy of a training with $Re = 100$ for initialization and four sub-domains was the only training able to run stable for 80 episodes, the final policy of this training was chosen for evaluation. As in section 5.33, here the training yielding the best seed value was taken.

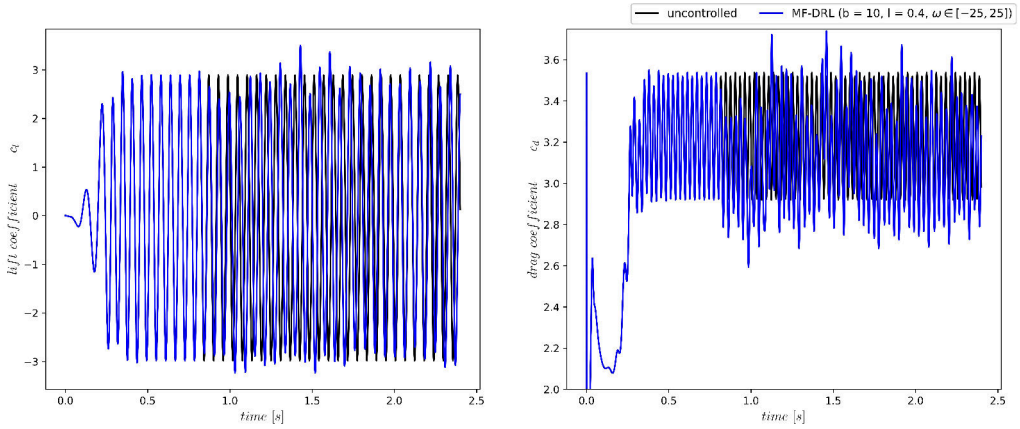


Figure 5.37: Lift and drag coefficients of the uncontrolled flow in comparison to the controlled one for a training with $l = 0.4$

Figure 5.37 shows the course of c_L and c_D of the uncontrolled and controlled flow using the final policy. It can be seen, that in contrast to simulations with $Re = 100$, the length of the transient phase is significantly decreased and takes here only approximately up to $t^* \approx 25$ instead of $t^* \approx 50$; but in order to ensure the comparability between simulations with different Reynolds numbers the control starts here at $t^* = 40$ as well. Further, the agent is not able to effectively reduce the loads acting on the cylinder, which indicates that the chosen setup for the training is not yet optimal. Table 5.4 shows the average c_L and c_D values of both cases in the quasi-steady state along with the corresponding standard deviations. As depicted, the policy is able to decrease the average c_D by 2.97% compared to the uncontrolled flow, however, the average c_L of the controlled flow is increased by 222.86% at the same time. Since the rewards depicted in fig. 5.36 showed a convergence until episode 80, it can be concluded that a further increase of the amount of episodes will not yield any measurable improvements with respect to the policy. Consequently, this indicates that a policy with $\omega \in [-25, 25]$ is not sufficient to effectively control the flow, although the rewards increased over the course of the training. A further increase of ω up to $\omega \in [-50, 50]$ is likely to improve the ability of flow control for $Re = 500$ as can be seen in fig. 5.34.

case	seed	$\mu(c_D)$	$\mu(c_L)$	$\sigma(c_D)$	$\sigma(c_L)$	$\mu(runtime)$
uncontrolled	–	3.233	−0.035	0.215	2.059	–
MF-DRL, $l = 0.4s$	1	3.137	−0.078	0.251	2.056	$12h : 01min$

Table 5.4: Average c_L and c_D values of the final policies in the quasi-steady state at $t = 1...1.6s$

Chapter 6

Conclusion

Climate change enforces the efforts to reduce the fuel consumption in aviation. While the efficiency of the propulsion system is one major concern, drag reduction of the aircraft itself using passive or active flow control showed a large potential. The applicability of active flow control methods, however, requires the definition of a robust control law, which is generally challenging. On the other hand, developments in AI and especially DRL over the past years showed the ability of learning optimal control laws in complex environments. The high computational costs associated with CFD limits the applicability of DRL for active flow control currently to rather simple cases, such as the well-known incompressible flow past a two-dimensional cylinder. Prior studies were able to successfully apply a variety of active flow control methods using DRL, which aimed to reduce the drag and lift forces acting on the cylinder. However, using exclusively CFD simulations in order to train the agent is computationally demanding and requires long runtimes as a consequence of the large amount of simulations which needs to be conducted. Model-based DRL on the other hand aims to approximate the environment with deep neural networks in order to decrease the runtimes, but ensuring a sufficient model accuracy over the course of the training is still a major difficulty. To proof the general feasibility of MB-DRL for active flow control, the model-free training routine for controlling the flow past a cylinder provided by the *drlfoam* framework was modified towards a MB-version in this thesis.

The derivation of requirements for a MB-training was conducted in a first step. Therefore, the current implementation of the MF-training was bench-marked with respect to the influence of the buffer size and trajectory length. It was found that the buffer size has no significant influence on rewards and runtimes, but an increase of the buffer size can be seen as beneficial since a higher buffer size provides a larger amount of training data for the models and sampling of initial states used for model-input. The remaining parameter trajectory length mainly influences the training stability and rewards. It was found that the trajectory length is required to be $l \geq 1s$ as a consequence of an initial transient phase, while for trajectory lengths of $l \geq 8s$ a convergence could have been observed. Consequently, trajectory lengths between $l = 2s$ and $l = 4s$ in combination with buffer sizes of $b = 8$ and $b = 10$ were seen as an achievable goal for a MB-algorithm.

In a next step, different approaches of modeling the CFD environment with deep neural networks were investigated. It was found that an alternation between the CFD environment followed by a model-training and several subsequent model-based episodes yielded best performance with respect to the prediction accuracy. Further, the course of the trajectories of c_L and the states showed a similar, periodic behavior while the trajectory of c_D was significantly more complex. This motivated to separate the predictions for c_L and the states from the predictions for c_D, which led to a further increase of the prediction accuracy. An optimization of architecture of the environment models, however, did only have a neglectable influence on the prediction accuracy.

After an integration of this approach into the PPO-training routine, the environment models were extended to a model-ensemble in order to reduce uncertainties within the prediction accuracy and to counteract a low-pass filtering behavior encountered when only one environment model was used. It was found that, independently of the trajectory length, $N_{models} = 5$ yielded the most stable training while achieving a good performance. The MB-training routine was subsequently optimized aiming to improve especially the stability of the training, however, the results and stability of the training were not always reproducible quantitatively when conducting a MB-training on different systems. Further, a high dependency on the initialization (seed value) of the training could have been observed, which increased with increasing trajectory length. The alternation between model-free and model-based episodes showed only little influence on the training stability, but indicated that with increasing trajectory length a more frequent switching to the CFD environment may be beneficial with respect to the training stability. A final benchmark of the MB-training for a Reynolds number of $Re = 100$ showed that the model-based training is able to run fairly stable, especially when shorter trajectories such as $l = 2s$ utilized, while yielding a comparable performance with respect to the ability of controlling the flow to the corresponding model-free training. Moreover, the model-based training was able to decrease the required runtime by up to 68.91% when conducting trainings on an HPC cluster. This decrease in runtime increases significantly when the trainings are conducted on a local machine, where the available computational resources are a limiting factor.

The capabilities of the environment models to generalize to flows with higher Reynolds numbers were investigated in a last step. Therefore, trainings with $Re = 500$ and $Re = 1000$ were conducted, but already the model-free training encountered here a variety of issues such as the stability of the training. The stability issues were partially caused by the necessary increase of ω to account for the higher Reynolds number and partially a consequence of increasing interprocess communication during the simulation due to the larger amount of sub-domains used. An initialization of the training at higher Reynolds numbers with the final policy of a corresponding training conducted at $Re = 100$ was able to mitigate these issues, but due to time constraints it was not possible to investigate the performance of DRL for active flow control at higher Reynolds numbers further. The usage of a random policy can consequently be considered as problematic when increasing the Reynolds number. Although MB-trainings were conducted using a random policy for initialization, the rewards showed no improvement over the course of the training, while the training itself was highly unstable. Conclusively, this behavior needs to be investigated more thoroughly in future work.

Current issues and outlook

The MB-training in general is able to run fairly stable while yielding good rewards. However, there is still a large potential for optimization and improvements, which shall be presented conclusively. The most important problem of the current implementation is the training stability, especially when increasing the trajectory length to $l \geq 4s$, CFD simulations tend to diverge leading to a crash of the PPO-training. These stability issues may be contributing to the different behavior with respect to the rewards when conducting MB-trainings on different systems, e.g. different HPC cluster. As a consequence of model uncertainties and a decreasing prediction accuracy with increasing trajectory length, the stability of the trainings as well as the performance with respect to the rewards are highly depended on initialization (seed value). The necessity of providing pre-trained policies in order to conduct stable trainings at higher Reynolds numbers lead further to high computational costs, especially when considering parameter studies, e.g. the influence of ω of rewards since for each configuration, first a MF-training at a low Reynolds number needs to be conducted prior running trainings for the actual Reynolds number of interest.

The causes of these issues can be divided mainly into two parts. The first part is the hardware

dependency and utilized libraries such as *OpenMPI*. It could been shown in section 5.4, that the number of MPI ranks (sub-domains) the computational domain is decomposed to has an influence on the training stability when the training is conducted on the *Phoenix* cluster of TU Braunschweig. It therefore needs to be further investigated if, e.g. this problem occurs on other systems such as the *AWS* HPC or on a local machine and in which boundaries the number of sub-domains can be chosen to ensure a stable training. Another aspect is the dependency of available resources, since it was found in section 5.2.5, that the stability of the training differs across different systems.

The second part concerns the MB-DRL algorithm itself, especially the improvement of the model accuracy may yield great potential regarding training stability and performance. There is always the possibility to implement more sophisticated models and algorithms, e.g. probabilistic or Bayesian models, however, when considering the current implementation a next important step may be a more thorough investigation of the influence of the model architecture. As presented in section 5.2.5, the influence of model architecture on results and stability is high, especially when considering no fully connect models.

The training stability may further be improved when the alternation between MB- and MF-episodes takes place dynamically instead of switching e.g. every fifth episode to the CFD simulation, maybe dependent on the current episode or based on other parameters such as the variance within the beta-distribution of the policy network. Further, up to this point the size of the model-ensemble for predicting $c_L - p$ is the same as for the ensemble predicting c_D, however, as discussed in section 5.1.5, with increasing N_{models} the ability to control the flow with respect to c_L deteriorates while the ability for c_D improves. This behavior may be exploited by using different N_{models} for the $c_L - p$ ensemble than for c_D ensemble.

Lastly, the loading of previous CFD episodes for model training and sampling of initial states can be optimized, especially the exception handling when CFD simulations diverge over a long period of episodes. On the other hand, the model-generated trajectories are only utilized once for updating the policy and then discarded for the remaining training. This inefficient usage of data maybe optimized by using improved correction models, which would be able to convert the model-generated trajectories in a way that they can be used for model-training as well.

Bibliography

[1] Abbeel, P., Quigley, M., and Ng, A. Y.: "Using inaccurate models in reinforcement learning". In: *Proceedings of the 23rd international conference on Machine learning*. Ed. by Cohen, W. ACM Other conferences. New York, NY: ACM, 2006, pp. 1–8. DOI: 10.1145/1143844.1143845.

[2] Beck, N., Landa, T., Seitz, A., et al.: "Drag Reduction by Laminar Flow Control". In: *energies* 11(1) (2018), pp. 1–28. DOI: 10.3390/en11010252.

[3] Belousov, B., Abdulsamad, H., Klink, P., et al., eds.: *Reinforcement learning algorithms: Analysis and applications*. Vol. 883. Studies in computational intelligence. Cham, Switzerland: Springer Nature, 2021. DOI: 10.1007/978-3-030-41188-6.

[4] Chua, K., Calandra, R., McAllister, R., et al.: "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *arXiv* (2018).

[5] Clavera, I., Rothfuss, J., Schulman, J., et al.: "Model-Based Reinforcement Learning via Meta-Policy Optimization". In: *arXiv* (2018).

[6] Darshan Thummar: "Active flow control in simulations of fluid flows based on deep reinforcement learning". MA thesis. Braunschweig: TU Braunschweig, 2021. DOI: 10.5281/ZENODO.4897961.

[7] Dong, H., Ding, Z., and Zhang, S., eds.: *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Singapore: Springer Singapore Pte. Limited, 2020.

[8] Elhawary, M. A.: "Deep Reinforcement Learning for Active Flow Control around a Circular Cylinder Using Unsteady-mode Plasma Actuators". In: *arXiv* (2020).

[9] Fabian Gabriel: "Aktive Regelung einer Zylinderumströmung bei variierender Reynoldszahl durch bestärkendes Lernen". MA thesis. Braunschweig: TU Braunschweig, 2021. DOI: 10.5281/ZENODO.5634050.

[10] Fan, D., Yang, L., Wang, Z., et al.: "Reinforcement learning for bluff body active flow control in experiments and simulations". In: *Proceedings of the National Academy of Sciences of the United States of America* 117(42) (2020), pp. 26091–26098. DOI: 10.1073/pnas.2004939117.

[11] Gabriel, E., Fagg, G. E., Bosilca, G., et al.: "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation". In: *Kranzlmüller, D., Kacsuk, P., Dongarra, J. (eds) Recent Advances in Parallel Virtual Machine and Message Passing Interface. EuroPVM/MPI 2004.* Vol. 3241, pp. 97–104. DOI: 10.1007/978-3-540-30218-6_19.

[12] Ganin, Y., Ustinova, E., Ajakan, H., et al.: "Domain-Adversarial Training of Neural Networks". In: *Journal of Machine Learning Research* 17 (2016), pp. 1–35.

[13] Garnier, P., Viquerat, J., Rabault, J., et al.: "A review on deep reinforcement learning for fluid mechanics". In: *Computers & Fluids* 225 (2021). DOI: 10.1016/j.compfluid.2021.104973.

[14] Haarnoja, T., Zhou, A., Abbeel, P., et al.: "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *arXiv* (2018).

[15] Hanks, G. W. and et. al.: *Natural Laminar Flow Airfoil analysis and trade studies*. Ed. by National Aeronautics and Space Administration. Hampton, Virginia.

[16] Hanna, B. N., Dinh, N. T., Youngblood, R. W., et al.: "Coarse-Grid Computational Fluid Dynamic (CG-CFD) Error Prediction using Machine Learning". In: *arXiv* (2017).

[17] Houthooft, R., Chen, X., Duan, Y., et al.: "VIME: Variational Information Maximizing Exploration". In: *arXiv* (2017).

[18] Janner, M., Fu, J., Zhang, M., et al.: "When to Trust Your Model: Model-Based Policy Optimization". In: *arXiv* (2019).

[19] Kalweit, G. and Boedecker, J.: *Uncertainty-driven imagination for continuous deep reinforcement learning*. 2017.

[20] Kingma, D. P. and Ba, J.: "Adam: A Method for Stochastic Optimization". In: *3rd International Conference for Learning, San Diego*.

[21] Kurutach, T., Clavera, I., Duan, Y., et al.: "Model-Ensemble Trust-Region Policy Optimization". In: *arXiv* (2018).

[22] Loshchilov, I. and Hutter, F.: *Decoupled Weight Decay Regularization*.

[23] Novati, G., Laroussilhe, H. L. de, and Koumoutsakos, P.: "Automating Turbulence Modeling by Multi-Agent Reinforcement Learning". In: *arXiv* (2020).

[24] OpenFOAM: *User Guide: Flow around a cylinder, OpenFOAM documentation*. URL: `https://www.openfoam.com/documentation/tutorial-guide/2-incompressible-flow/2.2-flow-around-a-cylinder`. Last access: 09.11.2022.

[25] OpenFOAM: *User Guide: incompressible solver algorithm, OpenFOAM documentation*. URL: `https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-incompressible-pimpleFoam.html`. Last access: 09.11.2022.

[26] Paris, R., Beneddine, S., and Dandois, J.: "Robust flow control and optimal sensor placement using deep reinforcement learning". In: *arXiv* (2020).

[27] Paszke, A., Gross, S., Massa, F., et al.: "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *arXiv* (2019).

[28] Pathak, J., Mustafa, M., Kashinath, K., et al.: "Using Machine Learning to Augment Coarse-Grid Computational Fluid Dynamics Simulations". In: *arXiv* (2020).

[29] PyTorch Foundation: *PyTorch documentation*. URL: `https://pytorch.org/docs/1.12/`. Last access: 09.11.2022.

[30] Qin, S., Wang, S., Rabault, J., et al.: "An application of data driven reward of deep reinforcement learning by dynamic mode decomposition in active flow control". In: *arXiv* (2021).

[31] Rabault, J., Kuchta, M., Jensen, A., et al.: "Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control". In: *Journal of Fluid Mechanics* 865 (2019), pp. 281–302. DOI: `10.1017/jfm.2019.62`.

[32] Rabault, J. and Kuhnle, A.: "Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach". In: *arXiv* (2019). DOI: `10.1063/1.5116415`.

[33] Raff, E.: *Inside deep learning: Math, algorithms, models*. Shelter Island: Manning Publications, 2022.

[34] Raissi, M., Perdikaris, P., and Karniadakis, G. E.: "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. DOI: `10.1016/j.jcp.2018.10.045`.

[35] Ríos Insua, D.: *Bayesian analysis of stochastic process models*. Wiley series in probability and statistics. Chichester, U.K.: Wiley, 2012.

[36] Schäfer, M., Turek, S., Durst, F., et al.: "Benchmark Computations of Laminar Flow Around a Cylinder". In: *Flow Simulation with High-Performance Computers II*. Ed. by Hirschel, E. H. Vol. 48. Notes on Numerical Fluid Mechanics (NNFM). Wiesbaden: Vieweg+Teubner Verlag, 1996, pp. 547–566. DOI: `10.1007/978-3-322-89849-4_39`.

[37] Schulman, J.: "Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs". MA thesis. EECS Department, University of California, Berkeley, 2016.

[38] Schulman, J., Levine, S., Moritz, P., et al.: "Trust Region Policy Optimization". In: *arXiv* (2015).

[39] Schulman, J., Moritz, P., Levine, S., et al.: "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *arXiv* (2015).

[40] Schulman, J., Wolski, F., Dhariwal, P., et al.: "Proximal Policy Optimization Algorithms". In: *arXiv* (2017).

[41] Schulze, E.: "Model-based Reinforcement Learning for Accelerated Learning From CFD Simulations". MA thesis. Braunschweig: TU Braunschweig, 2022. DOI: 10.5281/ZENODO.6375575.

[42] Shen, J., Zhao, H., Zhang, W., et al.: "Model-based Policy Optimization with Unsupervised Model Adaptation". In: *arXiv* (2020).

[43] Snoek, J., Larochelle, H., and Adams, R. P.: "Practical Bayesian Optimization of Machine Learning Algorithms". In: *arXiv* (2012).

[44] Sutton, R. S. and Barto, A. G.: *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[45] Tang, H., Rabault, J., Kuhnle, A., et al.: "Robust active flow control over a range of Reynolds numbers using an artificial neural network trained through deep reinforcement learning". In: *Physics of Fluids* 32(5) (2020). DOI: 10.1063/5.0006492.

[46] The Open MPI Project: *Open MPI documentation*. URL: https://www.open-mpi.org/doc/. Last access: 09.11.2022.

[47] Tokarev, M., Palkin, E., and Mullyadzhanov, R.: "Deep Reinforcement Learning Control of Cylinder Flow Using Rotary Oscillations at Low Reynolds Number". In: *Energies* 13(22) (2020), p. 5920. DOI: 10.3390/en13225920.

[48] Tzeng, E., Hoffman, J., Saenko, K., et al.: "Adversarial Discriminative Domain Adaptation". In: *arXiv* (2017).

[49] Varela, P., Suárez, P., Alcántara-Ávila, F., et al.: "Deep reinforcement learning for flow control exploits different physics for increasing Reynolds-number regimes". In: *arXiv* (2022).

[50] Wang, T., Bao, X., Clavera, I., et al.: "Benchmarking Model-Based Reinforcement Learning". In: *arXiv* (2019).

List of Figures

1.1 Rotation of the cylinder during the flow simulation [47] . . . 3
1.2 Comparison of the uncontrolled flow with the controlled flow at $Re = 100$ using model-free DRL . . . 3

2.1 Principle of reinforcement learning . . . 6
2.2 Pseudo-code of the PPO-algorithm . . . 7
2.3 Clipping of advantages . . . 7
2.4 Performance of TRPO and VIME on the *Walker2D* locomotion task . . . 8
2.5 Pseudo-code of the ME-TRPO algorithm . . . 9

3.1 Geometry of the numerical domain . . . 11
3.2 Spatial discretization of the flow problem . . . 12
3.3 Differences within the correlation heat maps for different grid refinement levels . 12
3.4 Grid convergence study for $Re = 500$ . . . 13

4.1 Average rewards received with respect to the buffer size and trajectory length . . 17
4.2 Average runtimes with respect to the buffer size and trajectory length . . . 18
4.3 Optimum of the runtime and received rewards . . . 18
4.4 Rewards received throughout the training for buffer sizes $b = 2$ and $b = 10$ with respect to the trajectory length . . . 19
4.5 Total rewards received over the course of the training for buffer sizes $b = 2$ and $b = 10$ with respect to the trajectory length . . . 20
4.6 L_2-norm of the prediction error for the two different approaches, averaged over all episodes and trajectories within the buffer . . . 21
4.7 Real trajectory generated in the CFD environment in comparison to the predicted trajectory by the environment model using a global model for all episodes . . . 22
4.8 Real trajectory generated in the CFD environment in comparison to the predicted trajectory by the environment model using a model trained with trajectories of episode 3 and 4 . . . 23
4.9 Real trajectory generated in the CFD environment in comparison to the predicted trajectory by the environment model using a model trained with trajectories of episode 12 and 13 . . . 23
4.10 The trajectories of the probes, corresponding to fig. 4.7 and 4.9 . . . 24
4.11 L_2- and L_1-norm of the prediction error of c_D with respect to the model architecture . . . 25
4.12 L_2- and L_1-norm of the prediction error of c_L with respect to the model architecture . . . 25
4.13 Comparison of the L_2-norm of the prediction error when using one model to predict c_L, p_i and c_D vs. one model for predicting c_L and p_i and a separate model for predicting c_D . . . 26
4.14 Real trajectory generated in the CFD environment in comparison to the predicted trajectory by the environment model using two models . . . 26
4.15 The trajectories of the probes corresponding to fig. 4.14 . . . 27
4.16 Power spectral density of the trajectories for c_L and c_D for the controlled and uncontrolled flow . . . 28
4.17 L_2-norm of the prediction error depended on the number of time steps used as model input . . . 29
4.18 L_2-norm of the prediction error depended on the number of time steps used as model input . . . 29

4.19 L_2-norm of the prediction error for $N_{t,input} = 30$ with respect to the epoch number 30
4.20 L_2-norm of the prediction error when predicting the change of state instead of the new state . . . 31
4.21 Real trajectory generated in the CFD environment in comparison to the predicted trajectory, ds was not scaled . . . 31

5.1 Different amounts of episodes taken for training the environment models . . . 33
5.2 Corresponding variances of the Beta-distribution with respect to the training routine 34
5.3 Influence of the trajectory length on the performance for a model-based PPO-training . . . 36
5.4 c_L and c_D over the course of the MB-PPO training when using different trajectory lengths . . . 36
5.5 Results using best policy for MF case and last CFD episode for MB case . . . 37
5.6 Qualitative comparison of the model-generated trajectory with a trajectory generated within the CFD environment . . . 38
5.7 Power spectral density for ω and frequencies of c_L and c_D . . . 38
5.8 Power spectral density for c_L, c_D and ω with respect to $N_{t,input}$ . . . 39
5.9 Rewards received for an MB-training with $b = 10, l = 4$ and different N_{models} in comparison to the MF-training . . . 40
5.10 Qualitative comparison of the trajectories for c_L and c_D at episode 80, for $b = 10, l = 4$ and different N_{models} in comparison to the MF-training . . . 40
5.11 Prediction accuracy of the different models for episode 80, $l = 4s$ and $b = 10$ . . . 41
5.12 Influence of the number of models within the ensemble on the received rewards for $b = 10$ and $l = 4s$ . . . 43
5.13 Influence of the number of models within the ensemble on c_L and c_D for $b = 10$ and $l = 4$ . . . 44
5.14 Differences between the rewards received on an HPC cluster and on a local machine 45
5.15 Training and validation losses using the original training routine . . . 46
5.16 Training and validation losses using the new training routine . . . 47
5.17 Comparison of the predicted trajectories with the original (CFD) one, for both training routines using a trajectory length of $l = 2s$ . . . 48
5.18 Comparison of the predicted trajectories with the original (CFD) one, for both training routines using a trajectory length of $l = 6s$ . . . 48
5.19 Comparison of the rewards received throughout the training for different seeds . 49
5.20 Influence of the number of models within the ensemble on the received rewards, here for $b = 10, l = 2$ . . . 50
5.21 Prediction accuracy of the different models when using the new training routine for episode 80, $l = 4s$ and $b = 10$ . . . 50
5.22 Influence of the number of models within the ensemble on the received rewards using $b = 10$ and $l = 4$ . . . 51
5.23 Influence of the ratio between MB- and MF-episodes for MB-trainings with $b = 10$ and $l = 2$ . . . 52
5.24 Lift and drag coefficients with respect to the episode number for MB-trainings with $b = 10$ and $l = 2s$ and different ratios of MB/MF episodes . . . 52
5.25 Influence of the ratio between MB- and MF-episodes for MB-trainings with $b = 10$ and $l = 4s$ . . . 53
5.26 Rewards received for MB-trainings on the *Phoenix* cluster in comparison to *AWS* using $b = 10$ and $l = 2s$ . . . 54
5.27 Rewards received for MB-trainings on the *Phoenix* cluster in comparison to *AWS* using $b = 10$ and $l = 4s$ . . . 55
5.28 Location of the additional probes placed in the vicinity of the cylinder . . . 56
5.29 Prediction accuracy of the environment models with and without additional correction models . . . 57
5.30 Comparison of the rewards for MB-training using the different approaches . . . 57
5.31 Comparison of the rewards for MB-training with differently weighted ω and c_D . 58
5.32 Comparison of the rewards for the final MF- and MB-trainings at $Re = 100$ . . . 59
5.33 Lift and drag coefficients of the uncontrolled flow in comparison to the controlled one for a training with $l = 4$ . . . 60
5.34 Reward received for a MF-training with a Reynolds number of $Re = 500$ with respect to the chosen ω using a trajectory length of $l = 0.4s$ . . . 61

5.35 Reward received for a MF-training with a Reynolds number of $Re = 500$ with respect to the chosen trajectory length using $\omega \in [-25, 25]$ 62
5.36 Reward received for a MF- and MB-trainings with a Reynolds number of $Re = 500$ with respect to the policy used for initialization 62
5.37 Lift and drag coefficients of the uncontrolled flow in comparison to the controlled one for a training with $l = 0.4$ 64

A.1 Lift and drag coefficients for $Re = 500$ depending on the number of mesh cells . . 76
A.2 Grid convergence study for $Re = 1000$ 76
A.3 Average c_L-values of the MF-training with respect to the buffer size and trajectory length 77
A.4 Average c_D-values of the MF-training with respect to the buffer size and trajectory length 77
A.5 Rewards received throughout the PPO-training for buffer size of $b = 4$ and $b = 6$ using different trajectory lengths 77
A.6 Rewards received throughout the PPO-training for a buffer size of $b = 8$ and different trajectory lengths 78
A.7 Convergence behavior of the PPO-training when extending the training to 200 episodes 78
A.8 Beta-distribution of additional entropy applied with respect to the episode, corresponding to section 5.1.1 78
A.9 Power spectral density for p_i using the final policy of a MB-training, corresponding to section 5.1.3 79
A.10 Rewards received for an MB-training using a model-ensemble with $b = 10, l = 2$ and different N_{models} in comparison to the MF-training 79
A.11 Rewards received for an MB-training using a model-ensemble with $b = 10, l = 6$ and different N_{models} in comparison to the MF-training, corresponding to section 5.1.4 79
A.12 Influence of the network architecture on the overall MSE-loss of c_D received throughout the training using the new training routine 81
A.13 Influence of the network architecture on the overall MSE-loss of c_L received throughout the training using the new training routine 81
A.14 Influence of the network architecture on the overall MSE-loss of p_i received throughout the training using the new training routine 82
A.15 Rewards for the original network architecture in comparison to the optimal one . 82
A.16 Comparison of the different adjustments to the training routine using $b = 10$ and $l = 2$ 83
A.17 Received rewards with respect to the seed value for a MB-training on *AWS* with fully-connected models and $N_{probes} = 12$ using $b = 10$ and $l = 6$ 83
A.18 Comparison of the rewards for MB-training with differently weighted ω and c_D . 84
A.19 Lift and drag coefficients with respect to the episode for the final trainings with $Re = 100$ and $l = 4s$ 85
A.20 Lift and drag coefficients with respect to the episode for the final trainings with $Re = 100$ and $l = 2s$ 85
A.21 Lift and drag coefficients of the uncontrolled flow in comparison to the controlled one for a training with $l = 2s$ at $Re = 100$ 85
A.22 Reward received for a MF-training with a Reynolds number of $Re = 1000$ with respect to the chosen ω using a trajectory length of $l = 0.2s$ 86

List of Tables

3.1 Characteristics of the mesh and simulation used in this thesis 13

4.1 Values used for the min- max scaling of c_L, c_D, runtimes and the rewards 17

5.1 Average c_L and c_D values of the final policies in the quasi-steady state at $t = 6...8s$ 37
5.2 Average runtimes for a MF- and MB-training on different systems 44
5.3 Average c_L and c_D values of the final policies in the quasi-steady state at $t = 6...12s$ 60
5.4 Average c_L and c_D values of the final policies in the quasi-steady state at $t = 1...1.6s$ 64

A.1 Hyperparameter settings of the PPO-algorithm 80
A.2 Hyperparameter settings of the global and the episode-wise environment model predicting c_D, c_L and p_i 80
A.3 Hyperparameter settings of the optimized training routine 80

Appendix A

Appendix

A.1 Navier-Stokes equations for incompressible flow

Incompressible flow can be described with the Navier-Stokes equations as

$$\nabla * \mathbf{u}^* = \mathbf{0} \tag{A.1}$$

$$\frac{\partial \mathbf{u}^*}{\partial t} + (\mathbf{u}^* * \nabla^*)\mathbf{u}^* = -\nabla^* p^* + \frac{1}{Re}\nabla^{*2}\mathbf{u}^* \tag{A.2}$$

where eq. A.1 describes the conservation of mass and eq. A.2 the conservation of momentum within the computational domain. These equations are non-dimensionalized using characteristic quantities of the flow problems, namely

$$t^* = t\,\frac{u_\infty}{d} \tag{A.3}$$

$$\mathbf{u}^* = \frac{\mathbf{u}}{\mathbf{u}_\infty} \tag{A.4}$$

$$p^* = p\,\frac{d}{\mu u_\infty} \tag{A.5}$$

$$\nabla^* = d\nabla \tag{A.6}$$

where the * superscript denotes all non-dimensional quantities. The Reynolds number Re can further be expressed as

$$Re = \frac{d\mathbf{u}_\infty}{\nu} \tag{A.7}$$

A.2 Adjustments for higher Reynolds numbers

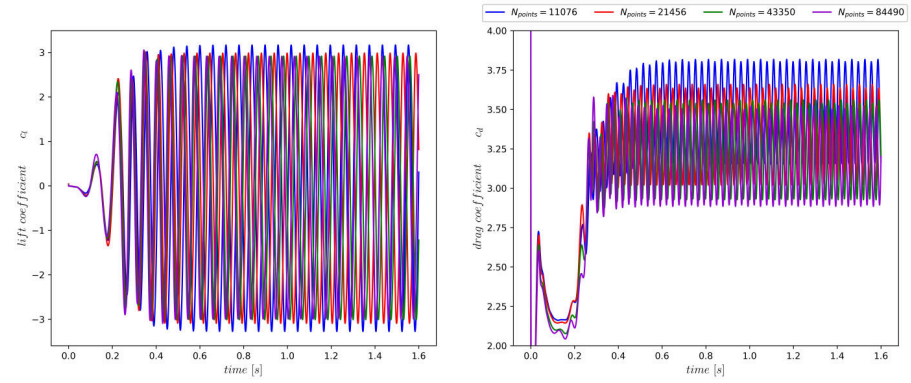


Figure A.1: Lift and drag coefficients for $Re = 500$ depending on the number of mesh cells

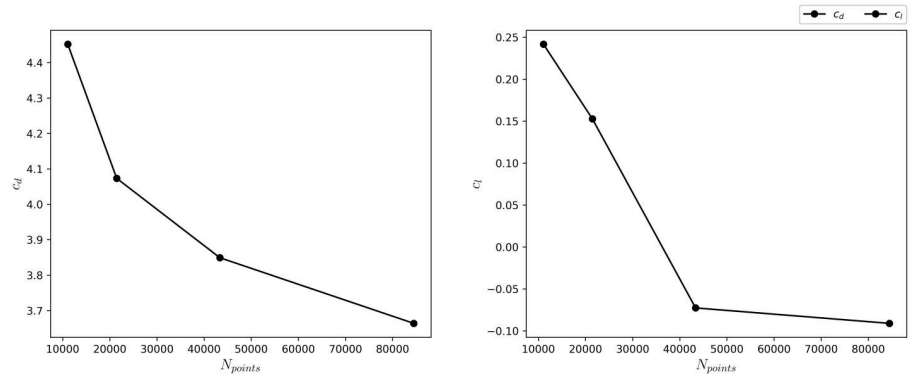


Figure A.2: Grid convergence study for $Re = 1000$

A.3 Influence of the buffer size and trajectory length (MF-DRL)

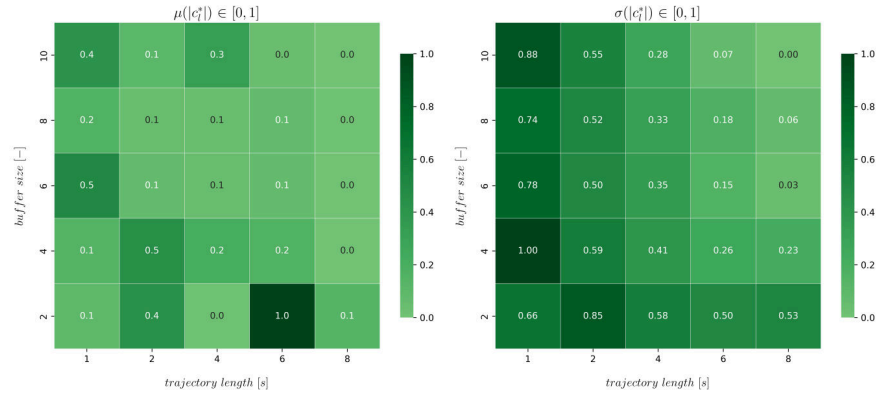


Figure A.3: Average c_L-values of the MF-training with respect to the buffer size and trajectory length

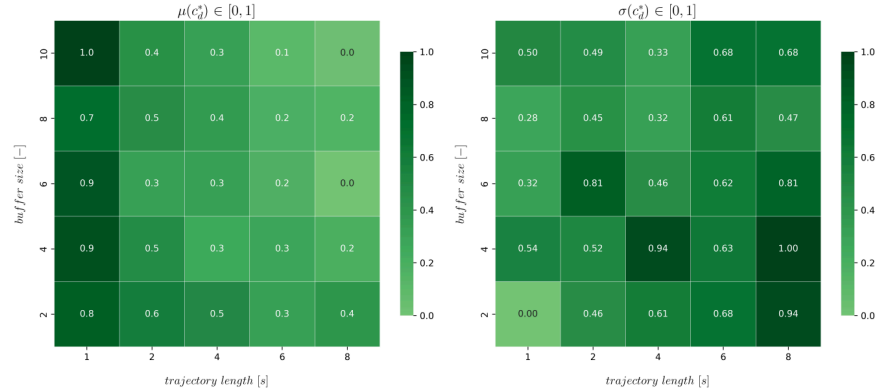
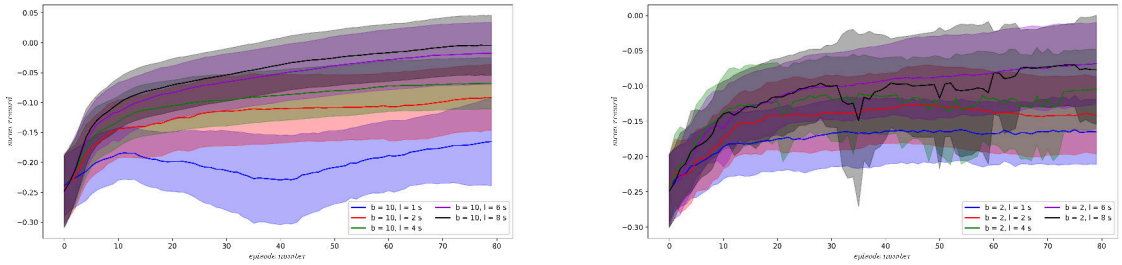


Figure A.4: Average c_D-values of the MF-training with respect to the buffer size and trajectory length



(a) Rewards for $b = 4$

(b) Rewards for $b = 6$

Figure A.5: Rewards received throughout the PPO-training for buffer size of $b = 4$ and $b = 6$ using different trajectory lengths

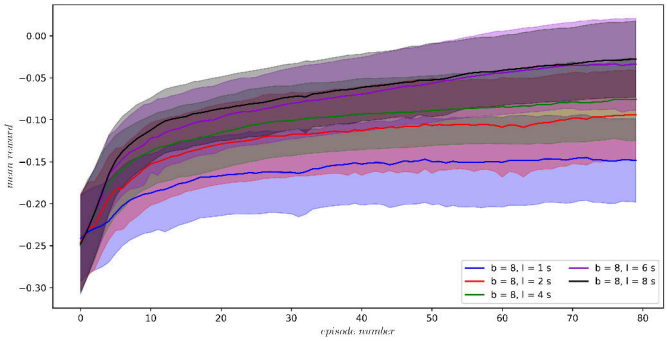


Figure A.6: Rewards received throughout the PPO-training for a buffer size of $b = 8$ and different trajectory lengths

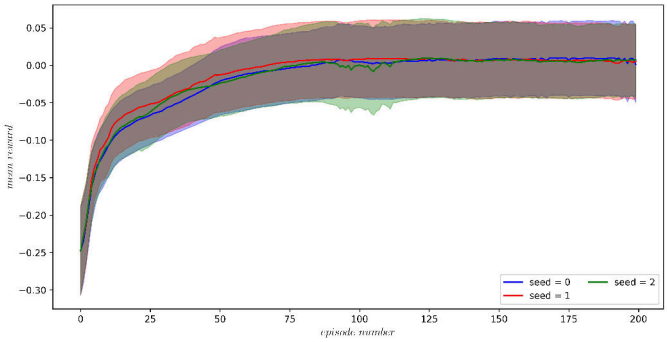


Figure A.7: Convergence behavior of the PPO-training when extending the training to 200 episodes using a buffer size of $b = 10$ and a trajectory length of $l = 8s$

A.4 Integration of the model-based approach into *drlfoam*

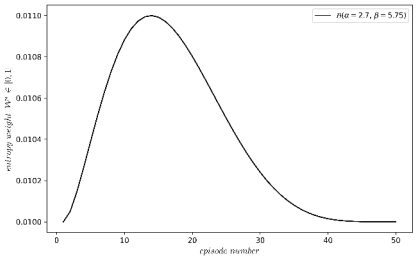


Figure A.8: Beta-distribution of additional entropy applied with respect to the episode, corresponding to section 5.1.1

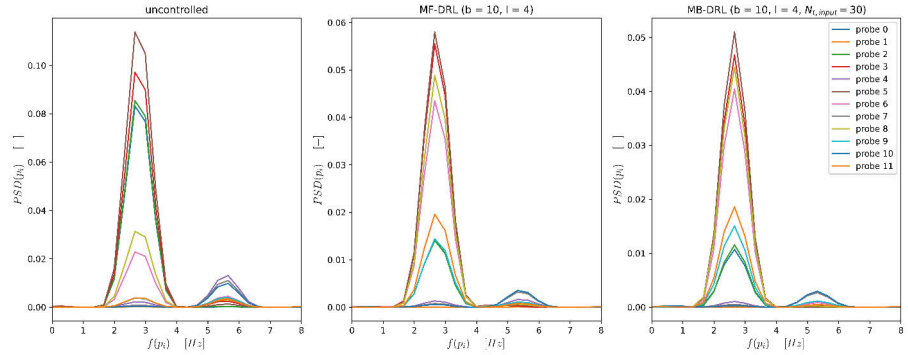


Figure A.9: Power spectral density for p_i using the final policy of a MB-training, corresponding to section 5.1.3

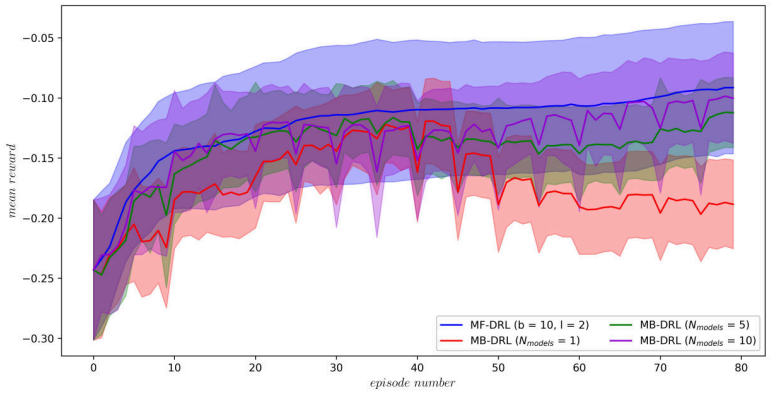


Figure A.10: Rewards received for an MB-training using a model-ensemble with $b = 10, l = 2$ and different N_{models} in comparison to the MF-training

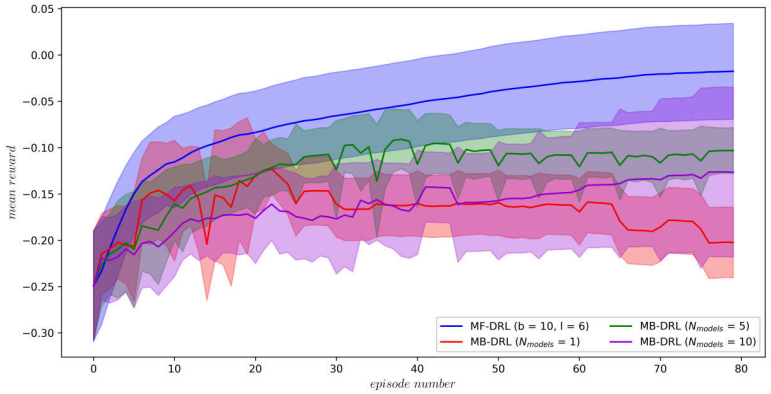


Figure A.11: Rewards received for an MB-training using a model-ensemble with $b = 10, l = 6$ and different N_{models} in comparison to the MF-training, corresponding to section 5.1.4

A.5 Hyperparameter settings

A.5.1 PPO

Parameter	Value network	Value network
N_{layers}	2	2
$N_{neurons}$	64	64
N_{epochs}	100	100
learning rate	0.001	0.0005

Table A.1: Hyperparameter settings of the PPO-algorithm, additionally $\gamma = 0.99$ and $\lambda = 0.97$

A.5.2 One global environment model vs. one new model each episodes

Parameter	global model	episode-wise model
$N_{t,input}$	2	2
N_{layers}	3	3
$N_{neurons}$	50	75
N_{epochs}	10000	10000
learning rate	0.0005	0.0005

Table A.2: Hyperparameter settings of the global and the episode-wise environment model predicting c_D, c_L and p_i

A.5.3 ME-MB-DRL

Parameter	global model	episode-wise model
$N_{t,input}$	30	30
N_{layers}	3	5
$N_{neurons}$	100	50
N_{epochs}	10000, 500	10000, 500
learning rate	0.001	0.001

Table A.3: Hyperparameter settings of the optimized training routine

A.6 Generalization of the training routine

A.6.1 Network architecture study new training routine

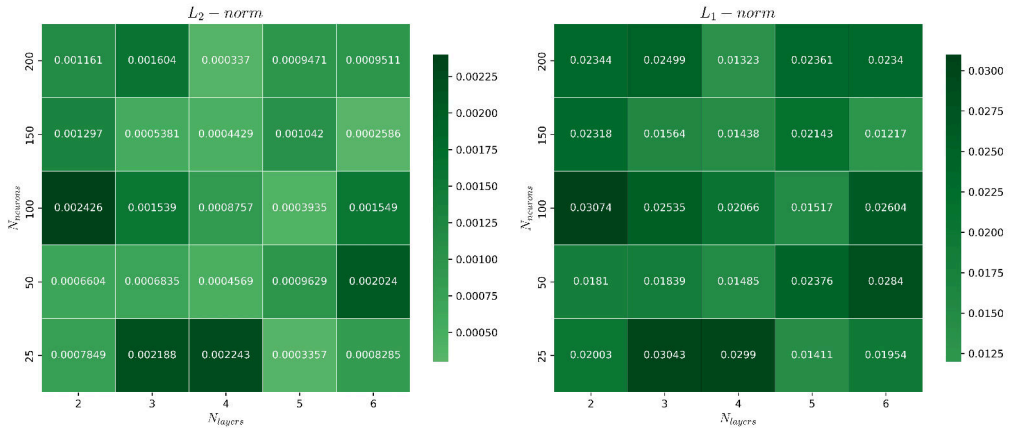


Figure A.12: Influence of the network architecture on the overall MSE-loss of c_D received throughout the training using the new training routine, here $b = 8, l = 2$ and $N_{models} = 5$. This figure corresponds to section 5.2.3.

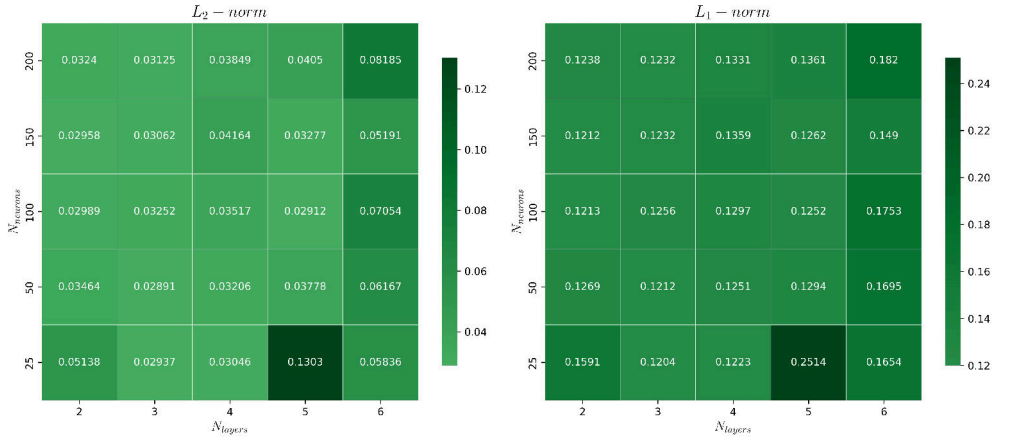


Figure A.13: Influence of the network architecture on the overall MSE-loss of c_L received throughout the training using the new training routine, here $b = 8, l = 2$ and $N_{models} = 5$. This figure corresponds to section 5.2.3 as well.

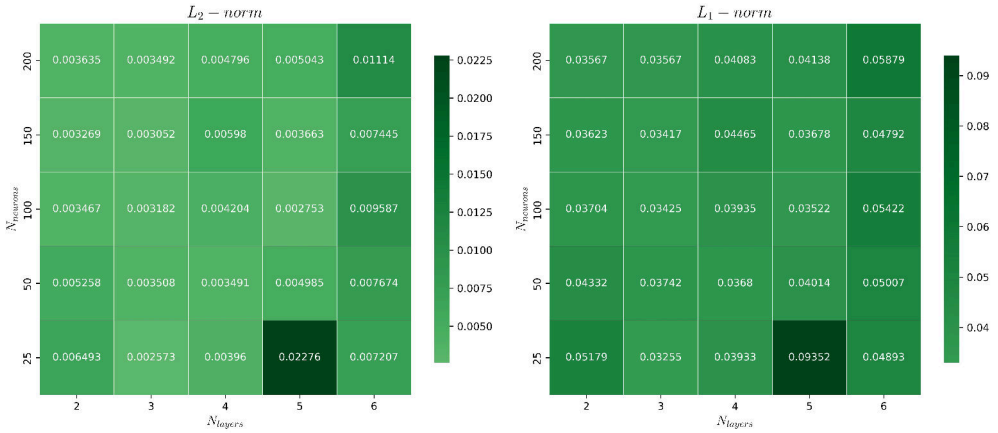


Figure A.14: Influence of the network architecture on the overall MSE-loss of p_i received throughout the training using the new training routine, here $b = 8, l = 2$ and $N_{models} = 5$. This figure also corresponds to section 5.2.3.

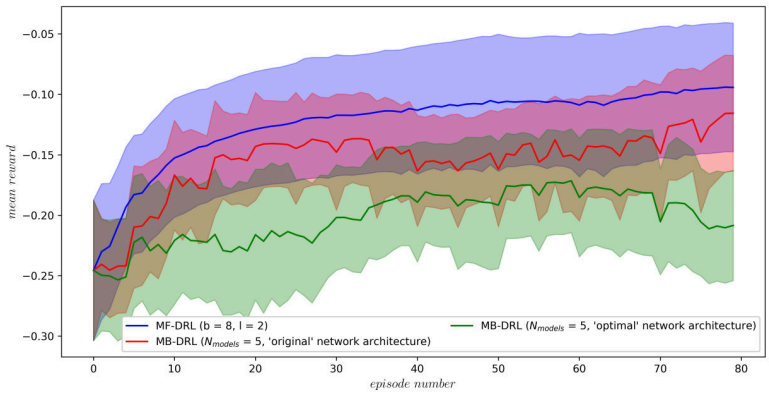


Figure A.15: Rewards for the original network architecture in comparison to the optimal one, here $b = 8, l = 2$ and $N_{models} = 5$. This figure corresponds to section 5.2.3.

A.6.2 Final improvements to the new training routine

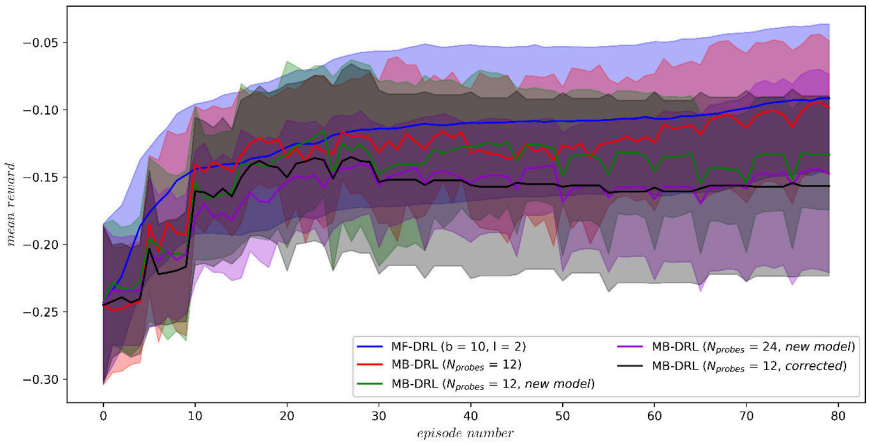


Figure A.16: Comparison of the different adjustments to the training routine using $b = 10$ and $l = 2$. This figure corresponds to section 5.2.5.

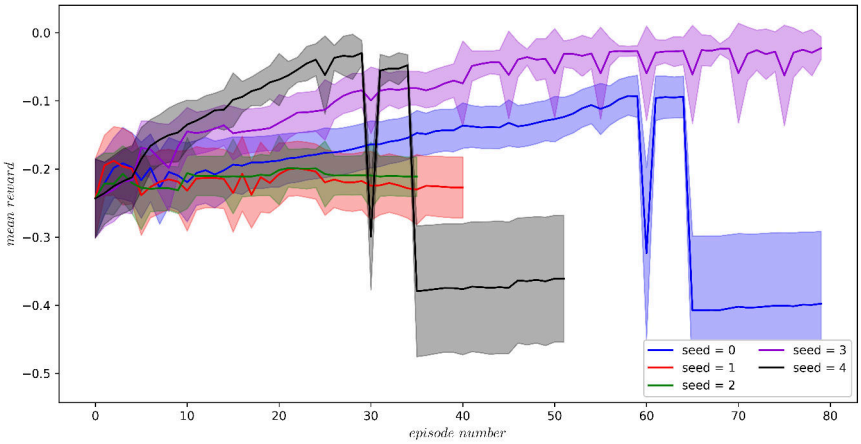


Figure A.17: Received rewards with respect to the seed value for a MB-training on *AWS* with fully-connected models and $N_{probes} = 12$ using $b = 10$ and $l = 6$. This figure corresponds to section 5.2.5.

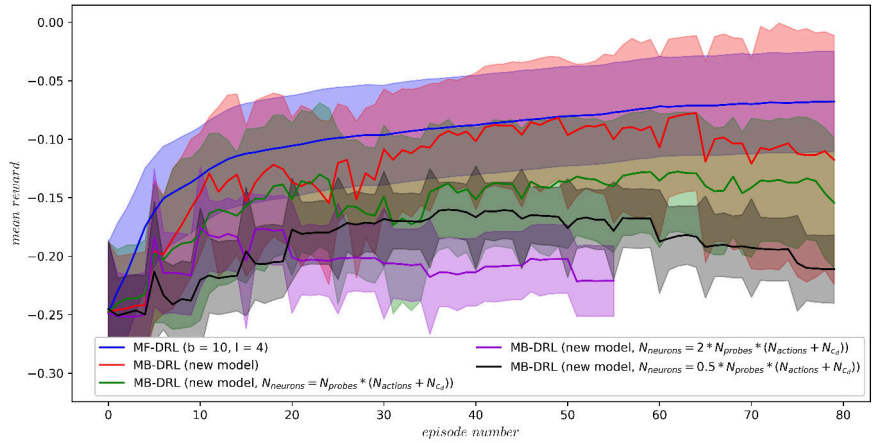


Figure A.18: Comparison of the rewards for MB-training with differently weighted ω and c_D, here for a training with $b = 10$ and $l = 4$. This figure corresponds to section 5.2.5.

A.7 Final results for a Reynolds number of Re = 100

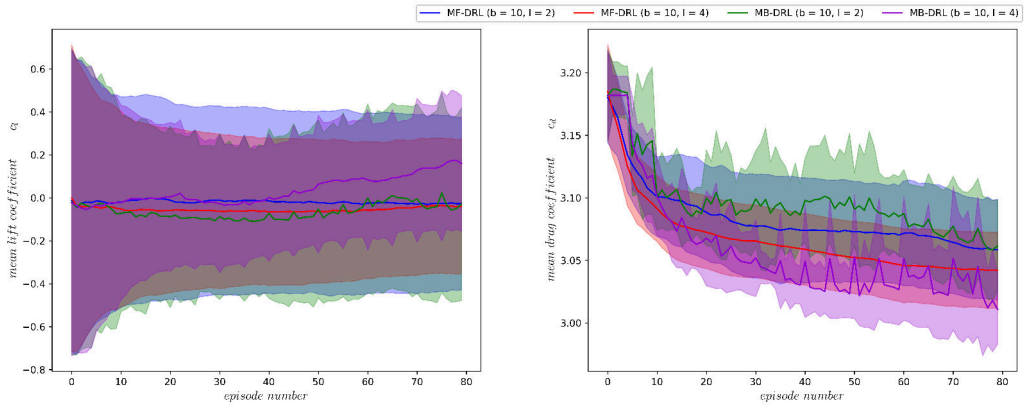


Figure A.19: Lift and drag coefficients with respect to the episode for the final trainings with $Re = 100$ and $l = 4s$. This figure corresponds to section 5.3.

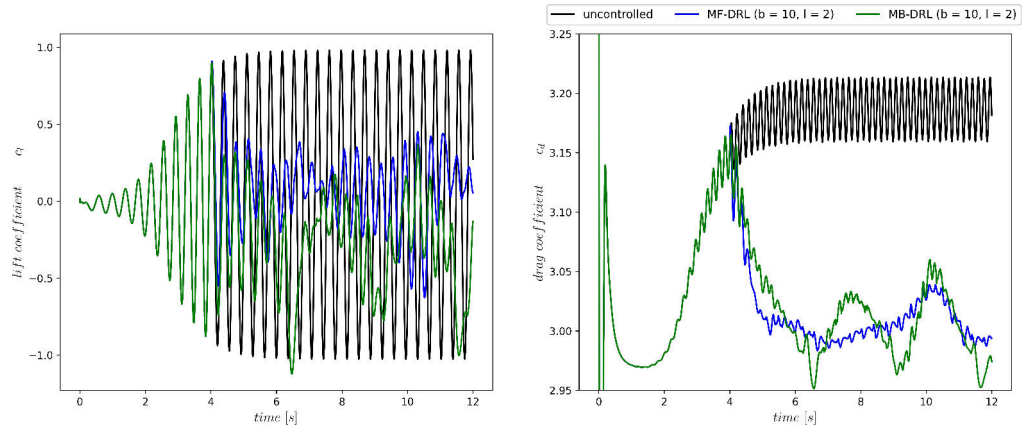


Figure A.20: Lift and drag coefficients with respect to the episode for the final trainings with $Re = 100$ and $l = 2s$. This figure corresponds to section 5.3.

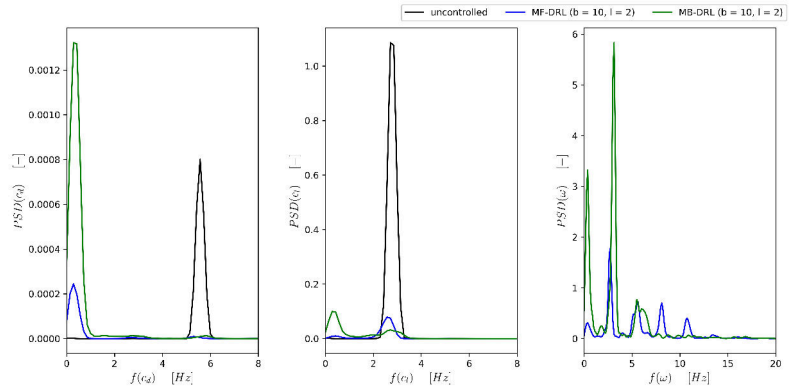


Figure A.21: Lift and drag coefficients of the uncontrolled flow in comparison to the controlled one for a training with $l = 2$ at $Re = 100$. This figure corresponds to section 5.3.

A.8 Model-based training at higher Reynolds numbers

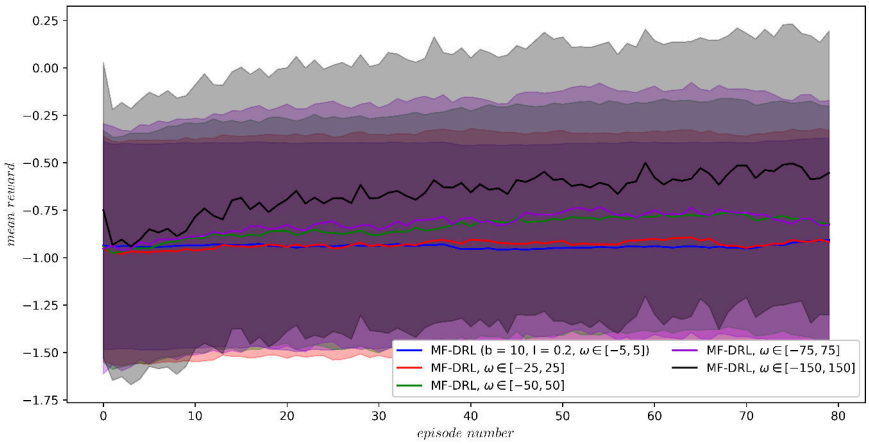


Figure A.22: Reward received for a MF-training with a Reynolds number of $Re = 1000$ with respect to the chosen ω using a trajectory length of $l = 0.2s$