Control of Two-way Coupled Fluid Systems with Differentiable Solvers

Brener Ramos, Felix Trost, Nils Thuerey
Technical University of Munich

Abstract

We investigate the use of deep neural networks to control complex nonlinear dynamical systems, specifically the movement of a rigid body immersed in a fluid. We solve the Navier Stokes equations with two way coupling, which gives rise to nonlinear perturbations that make the control task very challenging. Neural networks are trained in an unsupervised way to act as controllers with desired characteristics through a process of learning from a differentiable simulator. Here we introduce a set of physically interpretable loss terms to let the networks learn robust and stable interactions. We demonstrate that controllers trained in a canonical setting with quiescent initial conditions reliably generalize to varied and challenging environments such as previously unseen inflow conditions and forcing, although they do not have any fluid information as input. Further, we show that controllers trained with our approach outperform a variety of classical and learned alternatives in terms of evaluation metrics and generalization capabilities.

https://github.com/tum-pbs/two-way-coupled-control

1   Introduction

Control of tasks of physical systems are a ubiquitous challenge in science. In particular, fluids create very difficult environments which manifest themselves in simulations via the nonlinearities arising from the Navier Stokes (NS) equations. However, advancements in this field are important for society, and impact areas such as energy, transportation and biology (Barlas and Kuik 2007; Ho et al. 2003; Lord, Macmartin, and Tillman 2000).

Traditionally, open and closed loop control techniques have been investigated (Collis et al. 2004). The latter have clear advantages thanks to their conditioning on state measurements. We investigate and analyze a novel way to train closed loop controllers, namely via deep neural networks recurrently trained in a differentiable simulation environment with physics-based losses. This approach is motivated by the classical challenges of closed loop control for Navier-Stokes environments: fluid flows are complex and chaotic. Moreover, the number of degrees of freedom in numerical solvers is often very large, in turn requiring heavily reduced representations (Sipp and Schmid 2016; Noack, Tadmor, and Morzynski 2004; Bergmann and Cordier 2008; Proctor, Brunton, and Kutz 2016). Instead, training with a differen-

tiable simulator provides access to the full, unmodified physical environment, and provides reliable and diverse training feedback in the form of gradients.

More specifically, we investigate steering an actuated rigid body immersed in fluid systems with two way coupling, i.e. the rigid body influences the fluid around it and vice-versa. We focus on objectives that require the rigid body to reach specific target configurations, i.e. center of mass location and orientation. In this context, the differentiable simulations make it possible to learn controllers without providing ground truth control signals. Many works tackle fluids problems with reinforcement learning (Ma et al. 2018; Ren, Rabault, and Tang 2021; Paris, Beneddine, and Dandois 2021), which requires substantially longer training times and exhibit a high variance in performance when compared to our method.

An ubiquitous challenge for neural network approaches is generalization to conditions beyond the training distribution (Goodfellow, Bengio, and Courville 2016). We show that although training takes place in a quiescent flow condition, i.e. a fluid initially at rest, the networks trained via differentiable simulators are able to find control strategies that reliably handle more complex setups than those seen at training time. Their control characteristics are dictated by a set of physically interpretable loss terms, making it possible to favor desired aspects of the control, e.g., the amount of overshoot, tracking speed or maximum control effort. Our networks only receive relative directions in the form of displacement errors, rigid body velocities and previous control efforts. Therefore the neural networks act as a low-to-low dimensional mapping that uses easily accessible sensor data, which mimics potential real world applications. The performance of our networks is assessed in four different test scenarios with increasing levels of complexity. We show their advantages over a range of baseline algorithms, from linear controllers such as PID and loop shaping (McFarlane and Glover 1990; Kwakernaak 2002), to supervised and reinforcement learning algorithms (Haarnoja et al. 2018a).

2   Related Work

Many recent works have been investigating different ways of coupling control and deep learning. Since neural networks are good universal approximators (Hornik, Stinchcombe, and White 1989), many have investigated using them

as a reduced order model of a complex dynamical system (Eivazi et al. 2020; Hasegawa et al. 2020; Nair and Goza 2020), which can then be used as an inexpensive solver for known closed loop control techniques such as model predictive control (Bieker et al. 2020; Morton et al. 2018; Chen et al. 2021). Achieving linear-to-nonlinear mappings through learned Koopman operators has also been studied in recent works (Yeung, Kundu, and Hodas 2019; Li et al. 2020).

Another way of using deep learning for control purposes is through reinforcement learning (Verma, Novati, and Koumoutsakos 2018; Paris, Beneddine, and Dandois 2021; Ren, Rabault, and Tang 2021; Novati, Mahadevan, and Koumoutsakos 2019; Ma et al. 2018). In this case a neural network typically receives a representation of the multi dimensional state describing the system at a given time, e.g. velocity probes and scalar variables, and outputs the control efforts. This is achieved by training the network to maximize a reward function that describes a control objective. In recent years, a variety of refined reinforcement learning variants were proposed (Schulman et al. 2015; Ho and Ermon 2016; Schulman et al. 2017; Haarnoja et al. 2018b). These kind of algorithms traditionally require large amounts of data and training times, which is undesirable especially when considering computationally demanding simulation environments such as fluid simulations.

Recently, differentiable solvers were employed in numerous fields, such as robotics (Toussaint et al. 2019) and biology (Ingraham et al. 2019), were constructed to take advantage of deep learning tools via automatic differentiation. Since the gradients regarding a cost function are available, it is possible to directly solve for appropriate control efforts of a given task. This task could be placing a piece of cloth into a target container (Liang, Lin, and Koltun 2019), pouring liquids (Schenck and Fox 2018), moving a fluid to a specified region (Holl, Thuerey, and Koltun 2020) or generating a specified velocity field from an immersed body (Takahashi et al. 2021). To accomplish the control task, a full optimization needs to be performed for every timestep of a simulation to compute a suitable control signal. However, this is typically much too slow for practical applications with real time requirements. In this work we only use the gradients from the differentiable solver to train a network to act as a controller, which relies only on a sparse set of measurements from the environment. The resulting trained controller can then be evaluated very efficiently. Recent works also investigated using differentiable simulators to accelerate policy learning of various tasks, although generalization capabilities or robustness against disturbances were not assessed (Xu et al. 2022).

3 Methodology

3.1 Governing Equations

In physics and engineering the evolution of a physical system $\eta(x, t)$ is often described by a partial differential equation (PDE) as

$$\frac{\partial^n \eta}{\partial t^n} = \mathcal{F}\left(\eta, \frac{\partial \eta}{\partial x}, \dots, \frac{\partial^m \eta}{\partial x^m}, \frac{\partial \eta}{\partial t}, \dots, \frac{\partial^{n-1}\eta}{\partial t^{n-1}}, \omega(t, \eta)\right) \tag{1}$$

where $\mathcal{F}$ models the physical behavior of the system and $\omega(t, \eta)$ represents variables that influence it such as boundary conditions. If the system depends only on time as $\xi(t)$ then (1) reduces to an ordinary differential equation (ODE) as

$$\frac{\partial^n \xi}{\partial t^n} = \mathcal{G}\left(\xi, \frac{\partial \xi}{\partial t}, \dots, \frac{\partial^{n-1}\xi}{\partial t^{n-1}}, \omega(t)\right) \tag{2}$$

Given a generic control policy $\hat{\mathcal{P}}(t \mid \theta)$ parametrized by θ, an external actuation can be inserted into a system described by (2) according to

$$\frac{\partial^n \xi}{\partial t^n} = \mathcal{G}\left(\xi, \frac{\partial \xi}{\partial t}, \dots, \frac{\partial^{n-1}\xi}{\partial t^{n-1}}, \omega(t)\right) + \hat{\mathcal{P}}(t \mid \theta) \tag{3}$$

In this work, we target a coupled PDE-ODE system, interacting via boundary conditions ω and the control policy $\hat{\mathcal{P}}$. Integrating (3) over time yields a state modified by the policy, $\xi(t, \hat{\mathcal{P}})$, and the control task to reach ξ_{obj} is given by the minimization problem

$$\arg \min_\theta \|\xi_{obj} - \xi(t, \hat{\mathcal{P}}(t \mid \theta))\|. \tag{4}$$

More specifically, we use the incompressible Navier Stokes equations, which is a form of (1) with $n = 1$, that describes how a velocity field evolves given specified boundary conditions as the following

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u - \frac{\nabla p}{\rho} + \nu \nabla^2 u \tag{5}$$

where u is the velocity field, p is the pressure, ρ is the density and $\nu = \frac{\hat{u}\hat{L}}{Re}$ is the kinematic viscosity, where Re is the Reynolds number and $\hat{u}$ and $\hat{L}$ are a reference velocity and length, respectively. A Poisson equation is also solved for the pressure in order to enforce the velocity field to be divergence free.

We additionally target rigid objects immersed in the fluid. Their linear and angular movement can be described by (2) with $n = 2$ as

$$\frac{\partial^2 x_r}{\partial t^2} = \frac{1}{m} \sum F \tag{6}$$

$$\frac{\partial^2 \alpha}{\partial t^2} = \frac{1}{I} \sum T \tag{7}$$

where x_r is the body position, m the body mass, α the body angle and I its moment of inertia. The terms $\sum F$ and $\sum T$ denote the forces and torques that are acting upon the body, respectively. When using both (6) and (7) the system has 3 degrees of freedom (DOF). In a few cases below we will omit (7), yielding a simplified 2 DOF scenario. When dealing with a rigid body immersed in a fluid, these terms reduce to

$$\sum F = - \oint_S p \, \vec{n}(s) \, ds \tag{8}$$

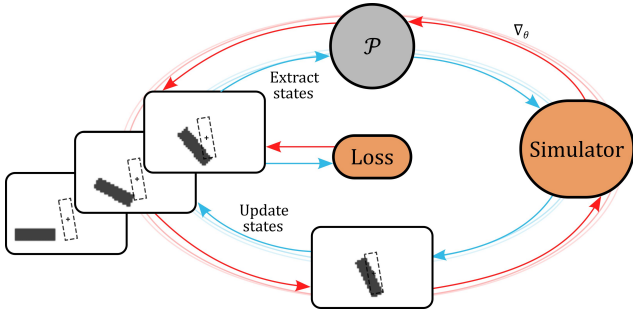$$\sum T = - \oint_S \vec{r}(s) \times p \, \vec{n}(s) \, ds \tag{9}$$

Figure 1: A schematic of the differentiable solver training: Blue arrows represent the forward pass while red ones illustrate the flow of gradients. Importantly, the loss signal is backpropagated through l simulation steps to provide policy $\mathcal{P}$ with long-term feedback about the flow environment.

where S is the body surface, $\vec{n}$ is the surface normal, $\vec{r}$ maps the surface location to the local coordinate system of the body, with the origin being the center of mass. Dirichlet boundary conditions for the NS simulation are imposed on the velocity field via $u_\Omega = \partial x_r / \partial t + \partial \alpha / \partial t \; r_\Omega$ at rigid body surface cells defined by contour Ω.

Together, (6) and (7) represent a coupled dynamical system subjected to nonlinear perturbations derived from the interaction between rigid body and fluid. Control efforts after exerted via forces:

$$\sum F = -\oint_S p \, \vec{n}(s) \, ds + F_c \qquad (10)$$

$$\sum T = -\oint_S \vec{r}(s) \times p \, \vec{n}(s) \, ds + T_c \qquad (11)$$

The specific control problem can then be formulated by finding the control efforts through $[F_c, T_c]^T = \mathcal{P}(t \mid \theta)$ so that

$$\arg\min_\theta \|e_{xy}\| + \|e_\alpha\| \qquad (12)$$

$$e_{xy} = x_{obj} - x_r(t, \mathcal{P}(t \mid \theta)) \qquad (13)$$

$$e_\alpha = \alpha_{obj} - \alpha(t, \mathcal{P}(t \mid \theta)) \qquad (14)$$

where x_{obj} and α_{obj} are an objective position and angle, respectively. Therefore the control task investigated can be summarized as controlling an ODE (rigid body movement equations) with highly nonlinear disturbances that emerge from a PDE (NS equations), which is also influenced by the ODE solution.

3.2 Differentiable Solver

Our differentiable fluid solver is based on Chorin-projections with a second-order advection operator. Signed distance functions of the moving obstacles are rasterized to the Eulerian simulation grid to flag cells as being either a fluid or an obstacle. Dirichlet BCs as well as isolating the rigid body surface are set in accordance to this cell classification in order to maintain differentiability. We implement our solver via the PhiFlow framework (https://github.com/tum-pbs/PhiFlow) using PyTorch as backend.

Each time step of the solver requires solving a Poisson's problem for the pressure considering the rigid body as boundary conditions. In order to avoid noisy gradients during backpropagation, the solver uses a custom PyTorch autograd function that solves a linear system in both backward and forward pass. The solid-fluid coupling is realized via a two-way coupling where NS and rigid body equations are solved in an alternating fashion. The body influences the fluid by changing the fluid's BCs, and the fluid acts on the rigid body by exerting force and torque calculated with (8) and (9). In this way, the control efforts gathered from the neural networks are propagated through the simulation graph, allowing gradients to flow from the loss function to the network weights.

4 Neural Networks as Control Policies

We investigate how to use neural networks to represent a policy $\mathcal{P}(z(t) \mid \theta)$ for the control task described by (12), where $z(t)$ is a set of discrete low dimensional state variables, which we denote with z for brevity. The network with weights θ acts as a policy, which receives the current and previous n_p states as input. It has the task to infer appropriate control efforts for a given learning objective. Each state is a low-dimensional array that consists of the spatial error e_{xy}, angular error e_α, rigid body linear velocity $\frac{\partial x_r}{\partial t}$, rigid body angular velocity $\frac{\partial \alpha}{\partial t}$ and control forces F_c and torques T_c. Also $\frac{\partial x_r}{\partial t}$ and $\frac{\partial \alpha}{\partial t}$ will be referred to as $\dot{x}$ and $\dot{\alpha}$, respectively. e_{xy}, $\dot{x}$ and F_c are expressed in the local reference frame of the rigid body. Therefore no global information describing the state of the fluid is transmitted to the control policy. We denote states at a discrete time t with a superscript ^{t}. In this way, the input for the networks at a time t can be expressed as $z = [w^t, w^{t-1}, \dots, w^{t-n_p}]$, where $w^t = [e_{xy}^t, \dot{x}^t, F_c^t, e_\alpha^t, \dot{\alpha}^t, T_c^t]^T$. In the following, a range of different learning procedures are investigated: training with differentiable physics simulators, a fully supervised approach, and a reinforcement learning variant.

4.1 Learning via a Differentiable Solver

Our method employs a fully differentiable solver which allows gradients to be provided to the neural network policy $\mathcal{P}_{\text{diff}}(z \mid \theta)$ about reactions of the physical system regarding previous policy actions and its temporal evolution. This policy can be trained without the need to pre-compute potentially sub-optimal training data. Rather, the network is left to discover the best possible policy over the course of the training in an unsupervised way. Our loss formulation includes a time horizon of l time steps as a central parameter (details on how we choose its value can be found in the appendix). The evaluations across this time interval leads to training signals that take into account how outputs of the policy network influence the future states of the environment. Via the differentiable solver, the loss signals are recurrently backpropagated to the policy, making it more "aware" of the consequences of previous actions. This process is illustrated in Figure 1. We make use of a loss function that combines three objectives. The objective term, O, typically dominates, and ensures that

the body reaches the target state:

$$O = \frac{\beta_{xy}}{l} \sum_{n=0}^{l-1} \|e_{xy}^n\|^2 + \frac{\beta_\alpha}{l} \sum_{n=0}^{l-1} \|e_\alpha^n\|^2, \qquad (15)$$

where the β are hyperparameters that weigh the different terms. However, a loss function with this term alone results in a controller with tendencies of overshoot since it does not account for the rigid body velocity. Hence we introduce a velocity term V:

$$V = \frac{\beta_{\dot{x}}}{l} \sum_{n=0}^{l-1} \frac{\|\dot{x}^n\|^2}{\beta_{prox}\|e_{xy}^n\|^2 + 1} + \frac{\beta_{\dot{\alpha}}}{l} \sum_{n=0}^{l-1} \frac{\|\dot{\alpha}^n\|^2}{\beta_{prox}\|e_\alpha^n\|^2 + 1} \qquad (16)$$

Far away from the target, larger spatial and angular errors in the denominators lead to smaller values of V. Closer to the target objective these errors approach zero, and hence V becomes an L2 norm of the linear and angular velocities. As a consequence the optimization guides the policy to slow down the body only when near the target, thus reducing overshooting effects. Following previous work (Bieker et al. 2020), we additionally include a term to avoid large control efforts as well as abrupt changes

$$E = \frac{\beta_F}{l} \sum_{n=0}^{l-1} \|F_c^n\|^2 + \frac{\beta_T}{l} \sum_{n=0}^{l-1} \|T_c^n\|^2 +$$

$$\frac{\beta_{\Delta F}}{l} \sum_{n=0}^{l-1} \|F_c^n - F_c^{n-1}\|^2 + \frac{\beta_{\Delta T}}{l} \sum_{n=0}^{l-1} \|T_c^n - T_c^{n-1}\|^2 \qquad (17)$$

Finally the combined loss function for the differentiable solver training can be written as

$$L = O + V + E \qquad (18)$$

Because of the direct physical impact of each hyperparameter, adjusting them is straight forward. For example, spatial tracking can be made more precise by increasing β_{xy} or the angular tracking can be accelerated by decreasing $\beta_{\dot{\alpha}}$. However, overly large values can result in overshooting. An ablation study can be found in Section 6.3. The hyperparameters for this work were chosen aiming for an overall balance of the objectives .

4.2 Supervised Learning

As a baseline for learning, we include a fully supervised learning approach. Due to a lack of optimal, ground-truth control policies, we construct a dataset in the following manner: we manually prescribe velocities to the rigid body so that it reaches an arbitrary target. We then compute the fluid forces acting upon the body and calculate the control efforts to cancel them and yield the acceleration of the prescribed trajectory. In this way, we obtain a set of states z with paired, expected control efforts $[\hat{F}_c(z), \hat{T}_c(z)]^T$. Training with these precomputed forces can be performed in a fully supervised way with the loss

$$L = \|[\hat{F}_c(z), \hat{T}_c(z)]^T - \mathcal{P}_{\text{sup}}(z \mid \theta)\|^2 \qquad (19)$$

We also investigated acquiring data by directly optimizing control efforts but this proved to be sub optimal. More details can be found in the appendix.

Table 1: Main parameters of experimental setups.

ID	Inflow	Buoy.	Forcing	DOF	Re
BaseNR	–	–	–	2	1000
BuoyNR	–	$\checkmark$	–	2	1000
Base	–	–	–	3	1000
Inflow	$\checkmark$	–	–	3	3000
InBuoy	$\checkmark$	$\checkmark$	–	3	3000
Hold	$\checkmark$	$\checkmark$	$\checkmark$	3	3000

4.3 Reinforcement Learning

Additionally, we include a reinforcement learning algorithm, Soft Actor Critic (SAC) (Haarnoja et al. 2018b), that works without making use of the solver gradients. SAC is a model-free variant that has the added benefit of being off-policy. Thus, past experiences can be stored in a replay buffer and are not invalidated by policy updates. As the simulation process is computationally very expensive, this increase in sample efficiency is highly beneficial. In our setting, the actor represents a control policy, and hence we refer to it as $\mathcal{P}_{\text{RL}}(z \mid \theta)$. The reward function uses the same formulation as the differentiable physics case, and is computed by multiplying equation (18) by -1. Other reward functions similar to the one from the original SAC algorithm were investigated but did not improve performance.

5 Experiments

We perform a series of experiments with increasing degrees of complexity to assess the generalization and stability capabilities and stability of the considered approaches. Below we explain the default parameters which are applicable unless noted otherwise. Details and deviating parameters for all experiments are provided in the appendix.

Datasets and Test Scenarios The training data consists of simulations with only one objective in the form of a target configuration, and uses the standard sets of parameters provided in Table 1. The two baseline versions are denoted by *BaseNR* , indicating "no rotation", for systems with 2 DOF, and *Base* for 3 DOF. All training samples use a quiescent flow ($u = 0$) as initial condition. Since the fluid is initially at rest, all perturbations are created from the rigid body movement. Validation datasets consist of 20 simulations with the same parameters as training, i.e. *BaseNR* and *Base* , but different objectives. To evaluate generalization of 2 DOF networks, simulations with parameters *BuoyNR* are performed, which increase the control task difficulty by introducing a lighter fluid source that disrupts the flow through buoyancy. For 3 DOF networks trained on *Base* , we use the environments *Inflow* , *InBuoy* and *Hold* . For *Inflow* , an inflow is present, the fluid is less viscous (higher Re) and Δt is smaller. Correspondingly, the controllers are sampled once every two timesteps. The environment *InBuoy* adds a lighter fluid source at the bottom of the domain that disrupts the flow through buoyancy. Finally, simulations with *Hold* parameters have additional prescribed forcing at certain moments. These test environments were designed to
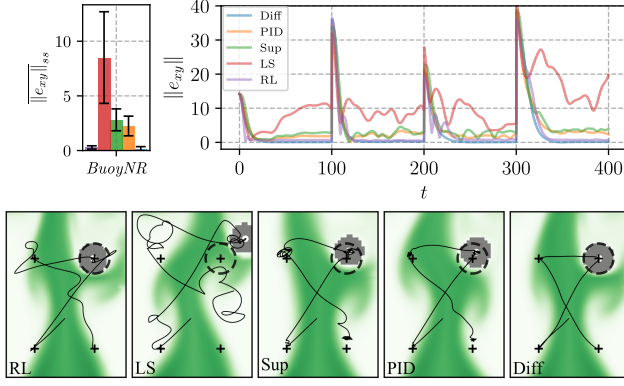
Figure 2: Average spatial steady state errors top-left), trajectories of one of the test simulations (bottom) and their error norms (top-right) with parameters *BuoyNR* . When compared to the other approaches $\mathcal{P}_{\text{diff}}$ exhibit less oscillations and a lower average steady state error.

deviate more and more strongly from the quiescent training conditions, making the control task progressively harder. These changes in environments are not directly transmitted to the controllers, rather, they should adapt to the new conditions based on their training. Unless stated otherwise, the test datasets are comprised of 5 simulations with different trajectories created by changing the objectives after $\Delta t = 100$ when considering *BuoyNR* , *Inflow* and *InBuoy* . Tests in the *Hold* environment are comprised of a single simulation which, in contrast to the other cases, has the goal to keep and stabilize the rigid body in the initial position. This is made more difficult by exerting additional forcing on the body. The supervised learning approach uses a separate, pre-computed dataset for training, which is comprised of 100 simulations from which 80 are used for training and 20 for validation and each one has 500 time steps.

Neural Network Representation and Training The underlying network architecture for all approaches is kept constant: two dense layers with ReLU activation, followed by a third dense layer. For the latter, the networks trained via differentiable physics and reinforcement learning have a hyperbolic tangent activation to ensure that the control efforts are bounded. As we achieved a better performance for the supervised case without the activation of the last layer, it was omitted there. Training the differentiable physics networks starts with a simulation with quiescent initial condition, gathering objectives from an uniform random distribution. After advancing the simulation so that n_p past states exist, the network training is activated and its outputs are used as control efforts. Once $l = 16$ time steps are available, we compute a loss and update θ. A discussion about how we choose l can be found in the appendix. After 1000 simulation steps we restart the simulation with new targets until n_i training iterations are performed. We choose $n_i = 1000$ and $n_i = 5000$ when training a network for 2 and 3 DOF systems, respectively. The reinforcement learning approach uses the same simulation environment as

the differentiable training but without making use of variables' gradients. After each simulated time step a batch of 128 samples is drawn from the replay buffer and used for training. In total, $n_i = 500,000$ iterations are performed and the validation error of intermediate models are computed. Furthermore, the model with the lowest validation error was chosen for further testing.

Evaluation Metrics As error metrics for comparing results we primarily use absolute errors in position and orientation, $||e_{xy}||$ and $||e_\alpha||$. We also compute an average steady state error $\overline{||e||}_{ss}$ that can assess the steady-state performance of the controllers without their initial transient phases as well as its standard deviation. It is calculated using the last $3/4$ of the time interval in which an objective was being tracked. However, since the rigid body starts in a null error position when considering tests in the *Hold* environment, the steady state error for these cases is the whole time average of position and orientation errors.

6 Results

6.1 Algorithmic Comparison

Baseline Algorithms Two types of linear controllers are tested as a reference to help assess the performance of the learned versions: a classic PID controller $\mathcal{P}_{\text{PID}}$ and a loop shaping controller $\mathcal{P}_{\text{LS}}$ designed through a blend of mixed-sensitivity design and the Glover-McFarlane method (Mc-Farlane and Glover (1990); Kwakernaak (2002)).

2 DOF Validation First, we consider a validation with the setup *BaseNR* (same simulation parameters as training but with different targets). The goal is to make sure all the controllers are functioning correctly for the conditions seen at training time. All the approaches considered are able to achieve a low error given sufficient time, which confirms they work as intended in conditions they were designed to operate in. Details and numerical values for all error measurements are given in the appendix.

Increased Difficulty Next we perform a test with the setup *BuoyNR* , which includes perturbations from a lighter fluid, different Re and a longer time window. The interaction between buoyancy and rigid body creates oscillatory flow structures and the control task becomes much more challenging. As a consequence, $\overline{||e_{xy}||}_{ss}$ is much larger for all approaches except for $\mathcal{P}_{\text{RL}}$ and $\mathcal{P}_{\text{diff}}$. The latter however has a more stable trajectory, as shown in Figure 2, and has a $\overline{||e_{xy}||}_{ss}$ of 0.19 ± 0.18 while the RL approach can achieve an error of 0.76 ± 0.26. A large amount of undesirable oscillations and overshooting is present when using all other controllers. It is worth noticing that $\mathcal{P}_{\text{RL}}$ requires many more iterations when compared to the other learned approaches. Additionally, $\mathcal{P}_{\text{RL}}$ training is inherently unstable and exhibits a high variance in performance, making it necessary to assess the performance of many intermediate models in order to find a well-performing candidate. Those factors result in $\mathcal{P}_{\text{RL}}$ taking 35 times longer to train than $\mathcal{P}_{\text{diff}}$. It can also be seen that the largest errors occur with $\mathcal{P}_{\text{LS}}$. For these reasons we will omit $\mathcal{P}_{\text{RL}}$ and $\mathcal{P}_{\text{LS}}$ for further tests.
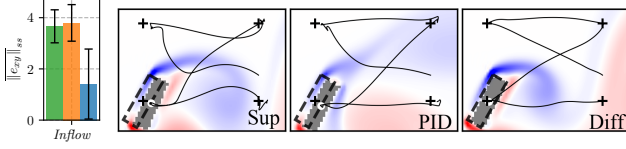
Figure 3: Average steady state errors (left), trajectories of one of the test simulations (right) with parameters *Inflow* . Trajectories are colored by vorticity ($t = 295$). $\mathcal{P}_{\text{sup}}$ and $\mathcal{P}_{\text{PID}}$ exhibit an offset when trying to reach target positions as well as mild oscillatory trajectories. Those behaviors are not observed when deploying $\mathcal{P}_{\text{diff}}$.

6.2 Increased Complexity and Generalization

We now evaluate the performance of the remaining methods for a rectangular body with rotation (3 DOF). The additional degree of freedom adds a significant amount of complexity to the control task, and we use a set of more complex test cases to evaluate generalization. In line with the 2 DOF case, we first perform an evaluation with validation cases with the same simulation parameters as training (*Base*). All three remaining methods fare well for these environments, and yield stable controllers. Additional details are provided in the appendix.

Generalization Tests First, tests with the *Inflow* setup are performed to assess the generalization capabilities of the considered controllers. Here the environments are changed substantially by introducing an inflow from the left side of the domain and utilizing a smaller viscosity (higher Re). The inflow together with the rigid body movement creates unsteady flow structures that vary depending on the box angle. $\mathcal{P}_{\text{diff}}$ is able to maintain a smaller $\overline{\|e_{xy}\|}_{ss}$ compared to the other controllers as can be seen in Figure 3. Very noticeable oscillations around the objective locations are present in the trajectory of $\mathcal{P}_{\text{sup}}$ and $\mathcal{P}_{\text{PID}}$. Instead, $\mathcal{P}_{\text{diff}}$ produces a stable orientation, showing that this policy successfully counteracts the perturbations caused by the strongly varying flow.

Inflow and Buoyancy: To increase difficulty, we introduce a source of lighter fluid that rises due to buoyancy at the bottom of the domain with parameters *InBuoy* . This test has the goal of assessing the robustness of the controllers further, since the fluid source tends to create higher frequency oscillations, which makes the control task harder. $\mathcal{P}_{\text{sup}}$ and $\mathcal{P}_{\text{PID}}$ show a considerable worsening of their performance with higher steady state errors and undesirable oscillations. On the other hand, $\mathcal{P}_{\text{diff}}$ maintains low values for $\overline{\|e_{xy}\|}_{ss}$ and $\overline{\|e_{\alpha}\|}_{ss}$ while successfully suppressing most of the perturbations as shown in Figure 4.

Hold: The last test introduces additional forcing to the rigid body in addition to the disturbances from the fluid. Since the simulation starts with the rigid body at the target position and orientation, the controllers primarily need to counter the external forces. $\mathcal{P}_{\text{diff}}$ is able to counteract the additional forcing, with the exception of a brief lapse at $t = 390$ as shown in Figure 5. This is caused by the fact that the fluid forces combined with additional forcing are
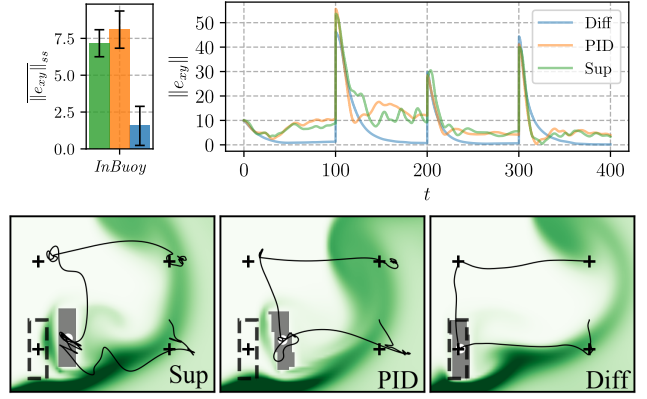

Figure 4: Average steady state errors (top-left), trajectories of one of the test simulations (bottom) and their error norms (top-right) with parameters *InBuoy* . Trajectories are colored by lighter fluid ($t = 195$). Performance of controllers are worsened but $\mathcal{P}_{\text{diff}}$ still maintains low steady state errors.
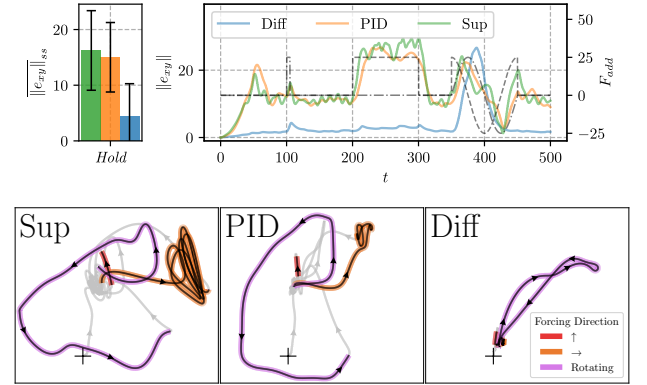

Figure 5: Average steady state errors (top-left), center-of-mass trajectories of simulations colored by forcing type (bottom) and their error norms (top-right) for the *Hold* environment. Additional forcing in x and y axes are represented by dashed and dotted-dashed black lines, respectively. Controllers must maintain the rigid body at the target location despite perturbations. $\mathcal{P}_{\text{diff}}$ is able handle vertical and horizontal forcing (red and orange) more efficiently than $\mathcal{P}_{\text{sup}}$ and $\mathcal{P}_{\text{PID}}$. Tracking briefly worsens when rotational forcing is applied (purple), due to maximum control efforts being smaller than the fluid forces combined with forcing.

momentarily larger than the maximum control forces. However $\mathcal{P}_{\text{diff}}$ is able to recover and lock into the target position again afterwards. Although $\mathcal{P}_{\text{sup}}$ and $\mathcal{P}_{\text{PID}}$ have unbounded maximum control efforts, they are not able to stabilize the rigid body at the target position. All steady state errors are summarized in Table 6 and Table 7 in the appendix.

6.3 Ablation Studies

We present an evaluation study with simulations performed with *InBuoy* parameters in order to assess the influence of
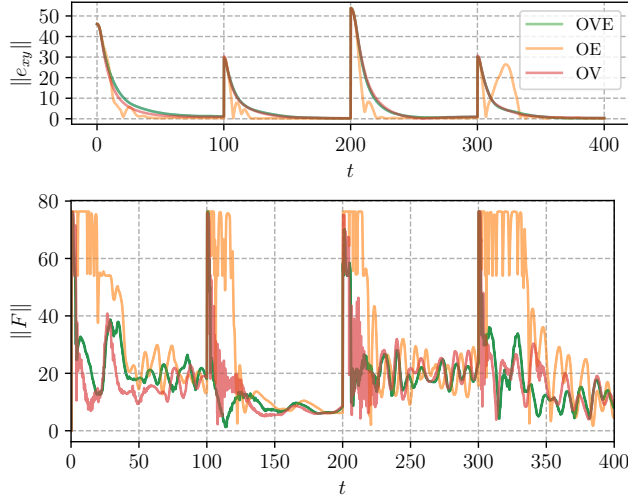
Figure 6: Errors norm (top) and control forces norm (bottom) on test with *InBuoy* parameters (bottom). All loss terms are necessary in order to accomplish steady error decay and maintain small and smooth control efforts.

the terms introduced in Section 4.1. One simulation is performed for each combination of loss terms. When using only the O term, the training is severely under constrained, and as a consequence the network learns to exert overly large forces. This destabilizes the simulations preventing a successful training. Therefore we compare the performance of networks trained with the combinations: *OVE* (all terms), *OV* (objective and velocity terms) and *OE* (objective and effort terms). Networks trained with *OVE* and *OV* exhibit similar error curves and trajectories as shown in Figure 6. The network trained with *OE* is also able to achieve low error values but it exhibits an uneven decay, which is an undesirable behavior. When analyzing the control efforts it can be seen that *OE* provides a network that produces maximum control efforts more often, which means it uses more energy. When considering *OV*, there are no constraints on control efforts and consequently it learns a control policy that modulates the control efforts with a high frequency - an undesirable behavior in practice.

6.4 Discussion

Taken together, the previous set of tests show the advantages of the proposed training via differentiable simulations. It is able to provide a neural network with feedback to control a dynamical system subjected to nonlinear perturbations for long periods of time, even though training is performed in a different, simplified environment. By backpropagating through l steps, the gradient of the time evolution of the system greatly helps the optimization process to find useful relations between the input measurements in order to generate appropriate controls while not relying on inputs containing global information. Our results show that complex control strategies can be learned in an unsupervised manner as long as the control objective and controller characteristics can be expressed in a mathematically meaningful way.

Another interesting characteristic is the robustness against the disturbances seen in the tests. Although some tracking performance is lost the more complex the tests become, $\mathcal{P}_{\text{diff}}$ is able to handle new situations without the strongly deteriorated tracking performance of the other controllers. Our tests indicate that this is a consequence of the differentiable solver, which provides a varied learning signal and results in a neural network that robustly handles a large variety of conditions.

It is also conceptually very appealing for control algorithms to be able to use the original set of equations of a system, even if they are as nonlinear as the NS equations. This is due to the fact that reduced representations often do not portrait nonlinear and unexpected behavior well enough. Instead, the differentiable simulation approach allows training the controller efficiently using the full set of model equations, such that it can adapt to the subtleties of the environment.

On the other hand, a drawback of the proposed approach is the fact that differentiable simulations lead can lead to increased training times because the simulation has to be evaluated in the forward, as well as the backward pass. In our experiments we found this can be up to 4 times slower than training via a supervised policy. The latter, however, induces an additional cost for generating the training data, which is likely to be substantial. The differentiable solver training shares the increased training cost with reinforcement learning, but a single iteration of the former is roughly twice as expensive due to the backwards path for calculating the gradient of the simulator. However, the gradients typically lead to faster convergence. In our tests, the reinforcement learning approach on average takes an order of magnitude longer to train and exhibits high variance in terms of models performance. Yet, the resulting neural network is equally fast to evaluate for all three cases, since it relies on a low-dimensional set of measurements as inputs. Hence the increased one-time cost to train with a differentiable simulator compared, e.g., to fully supervised training, can pay off significantly over the course of a large number of evaluations when the controller is deployed.

7 Conclusions

We have studied the use of differentiable simulations to train neural networks acting as controllers for complex dynamical systems. The considered system describes the movement of a rigid body subjected to nonlinear perturbations derived from the Navier Stokes equations, posing a very challenging control task. The proposed approach, which introduced a set of physically interpretable loss terms, is able to train robust controllers without the need to provide reference data for training while relying on a sparse set of measurements. It is able to produce controllers that generalizes very well to substantially different, challenging flow conditions. Numerous interesting venues for future research exist based on our results, such as exploring the transfer of synthetically trained controllers to real-world environments and using the proposed strategy to develop controllers for other control tasks using differentiable simulators.

References

Barlas, T.; and Kuik, G. 2007. State of the Art and Prospectives of Smart Rotor Control for Wind Turbines. *Journal of Physics: Conference Series*, 75: 012080.

Bergmann, M.; and Cordier, L. 2008. Optimal control of the cylinder wake in the laminar regime by trust-region methods and POD reduced-order models. *Journal of Computational Physics*, 227(16): 7813–7840.

Bieker, K.; Peitz, S.; Brunton, S. L.; Kutz, J. N.; and Dellnitz, M. 2020. Deep model predictive flow control with limited sensor data and online learning. *Theoretical and Computational Fluid Dynamics*, 34(4): 577–591.

Chen, K.; Lin, J.; Qiu, Y.; Liu, F.; and Song, Y. 2021. Deep learning-aided model predictive control of wind farms for AGC considering the dynamic wake effect. *Control Engineering Practice*, 116: 104925.

Collis, S. S.; Joslin, R. D.; Seifert, A.; and Theofilis, V. 2004. Issues in active flow control: theory, control, simulation, and experiment. *Progress in Aerospace Sciences*, 40(4): 237–289.

Eivazi, H.; Veisi, H.; Naderi, M. H.; and Esfahanian, V. 2020. Deep neural networks for nonlinear model order reduction of unsteady flows. *Physics of Fluids*, 32(10): 105104.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018a. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. PMLR.

Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. 2018b. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hasegawa, K.; Fukami, K.; Murata, T.; and Fukagata, K. 2020. Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes. *Theoretical and Computational Fluid Dynamics*, 34(4): 367–383.

Ho, J.; and Ermon, S. 2016. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29: 4565–4573.

Ho, S.; Nassef, H.; Pornsinsirirak, N.; Tai, Y.-C.; and Ho, C.-M. 2003. Unsteady aerodynamics and flow control for flapping wing flyers. *Progress in Aerospace Sciences*, 39(8): 635–681.

Holl, P.; Thuerey, N.; and Koltun, V. 2020. Learning to Control PDEs with Differentiable Physics. In *International Conference on Learning Representations*.

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

Ingraham, J.; Riesselman, A.; Sander, C.; and Marks, D. 2019. Learning Protein Structure with a Differentiable Simulator. In *International Conference on Learning Representations*.

Kwakernaak, H. 2002. Mixed Sensitivity Design. *IFAC Proceedings Volumes*, 35(1): 61–66. 15th IFAC World Congress.

Li, Y.; He, H.; Wu, J.; Katabi, D.; and Torralba, A. 2020. Learning Compositional Koopman Operators for Model-Based Control. In *International Conference on Learning Representations*.

Liang, J.; Lin, M.; and Koltun, V. 2019. Differentiable Cloth Simulation for Inverse Problems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Lord, W.; Macmartin, D.; and Tillman, T. 2000. Flow control opportunities in gas turbine engines.

Ma, P.; Tian, Y.; Pan, Z.; Ren, B.; and Manocha, D. 2018. Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(4): 1–11.

McFarlane, D. C.; and Glover, K. 1990. *Robust controller design using normalized coprime factor plant descriptions*, volume 138. Springer.

Morton, J.; Jameson, A.; Kochenderfer, M. J.; and Witherden, F. 2018. Deep Dynamical Modeling and Control of Unsteady Fluid Flows. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Nair, N. J.; and Goza, A. 2020. Leveraging reduced-order models for state estimation using deep learning. *Journal of Fluid Mechanics*, 897: R1.

Noack, B.; Tadmor, G.; and Morzynski, M. 2004. *Low-Dimensional Models for Feedback Flow Control. Part I: Empirical Galerkin Models*.

Novati, G.; Mahadevan, L.; and Koumoutsakos, P. 2019. Controlled gliding and perching through deep-reinforcement-learning. *Physical Review Fluids*, 4(9): 093902.

Paris, R.; Beneddine, S.; and Dandois, J. 2021. Robust flow control and optimal sensor placement using deep reinforcement learning. *Journal of Fluid Mechanics*, 913: A25.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

Proctor, J. L.; Brunton, S. L.; and Kutz, J. N. 2016. Dynamic Mode Decomposition with Control. *SIAM Journal on Applied Dynamical Systems*, 15(1): 142–161.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268): 1–8.

Ren, F.; Rabault, J.; and Tang, H. 2021. Applying deep reinforcement learning to active flow control in weakly turbulent conditions. *Physics of Fluids*, 33(3): 037121.

Schenck, C.; and Fox, D. 2018. SPNets: Differentiable Fluid Dynamics for Deep Neural Networks. In *Proceedings of the Second Conference on Robot Learning (CoRL)*. Zurich, Switzerland.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International conference on machine learning*, 1889–1897. PMLR.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347.

Sipp, D.; and Schmid, P. J. 2016. Linear Closed-Loop Control of Fluid Instabilities and Noise-Induced Perturbations: A Review of Approaches and Tools1. *Applied Mechanics Reviews*, 68(2). 020801.

Takahashi, T.; Liang, J.; Qiao, Y.-L.; and Lin, M. C. 2021. Differentiable Fluids with Solid Coupling for Learning and Control. In *AAAI*.

Toussaint, M.; Allen, K. R.; Smith, K. A.; and Tenenbaum, J. B. 2019. Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 6231–6235. International Joint Conferences on Artificial Intelligence Organization.

Verma, S.; Novati, G.; and Koumoutsakos, P. 2018. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23): 5849–5854.

Xu, J.; Macklin, M.; Makoviychuk, V.; Narang, Y.; Garg, A.; Ramos, F.; and Matusik, W. 2022. Accelerated Policy Learning with Parallel Differentiable Simulation. In *International Conference on Learning Representations*.

Yeung, E.; Kundu, S.; and Hodas, N. 2019. Learning Deep Neural Network Representations for Koopman Operators of Nonlinear Dynamical Systems. In *2019 American Control Conference (ACC)*, 4832–4839.

A Appendix

A.1 PID

The P gain was adjusted so that the maximum control effort from the PID controller is in the same order of magnitude as the one from the network controllers. The D gain is then tuned to be as low as possible while still avoiding overshoot and the same is done for the I gain. The gains for the controllers for each setup are displayed in Table 2 and the control effort U is obtained by

$$U^t = Pe^t + D\frac{e^t - e^{i-1}}{0.1} + 0.1\sum_{n=0}^{i} e^n I \qquad (20)$$

Table 2: Gains of PID controller.

	P	D	I
Cylinder - Forces	1	8	0.001
Box - Forces	2	15	0.001
Box - Torque	100	1000	0.01

A.2 Loop Shaping

Loop Shaping design consists of finding a controller K so that the open loop response $K * P$, where P is the system plant, behaves as close as possible to the open loop response of a transfer function P', which is chosen by the user.

A common choice for P' is $P'(s) = \frac{\omega_b}{s}$ where ω_b is the control bandwidth and s is a complex frequency. This function has the property of having high gains for low frequencies and low gains for high frequencies. In other words, an input signal with frequency higher than w_b (noise) will be dampened and an input signal with frequency lower than w_b (perturbations) will be amplified.

We use a combination of two loop-shaping methods: mixed-sensitivity-design, which favors performance, and the Glover-McFarlane method, which favors robustness to plant uncertainty, as implemented by the Matlab *loopsyn()* function. It features a parameter α: For $\alpha = 0$ the controller has the best performance and while $\alpha = 1$ favors robustness. From a range of experiments with our physical environment, we choose $\alpha = 0.95$ and $w_b = 0.2$. After converting the found controller from the Laplace domain to the discrete one, we obtain the coefficients shown in Table 3 and the control effort U is calculated according to

$$U^t = \sum_{p=0}^{2} e^{i-p}n_p - \sum_{p=1}^{2} U^{i-p}d_p \qquad (21)$$

where i is the index of the current time step, e is the error and n and d are the controller coefficients.

Table 3: Loop shaping coefficients.

p	n_p	d_p
0	$1.1700924033918623e00$	-
1	$-1.4694211940919182e00$	$-1.2306775904257603e00$
2	$3.0598060140064326e-01$	$2.6726488821832250e-01$

A.3 Networks Architecture Details

The dimensions of the considered neural network layers are given in Table 4, and its architecture is visualized in Figure 7. The network of the 2 DOF setup has a total of 2206 trainable parameters while the one used in the 3 DOF setup has 2243 trainable parameters.

Table 4: Input/output sizes of neural network layers.

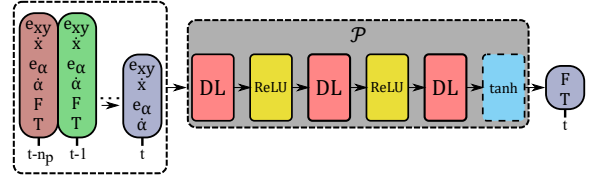| Layer	2 DOF	3 DOF
0	[16, 38]	[32, 32]
1	[38, 38]	[32, 32]
2	[38, 2]	[32, 3]

Figure 7: Visualization of the neural network architecture.

A.4 Training Details

Differentiable Physics The network training parameters for all setups investigated with differentiable physics can be found in Table 5. A learning rate of 0.01 was used in our tests. Additionally, learning rate decay is used so that the learning rate drops to half of its initial value every 200 and 1000 training iterations for the 2 DOF and 3 DOF setups, respectively. The networks inputs and outputs are normalized based on a set of measured simulation statistics.

Effect of Time Horizon l The effect of training with different time horizons l is also investigated. A test consisting of one simulation with parameters *InBuoy* and a N shaped trajectory is conducted and the performance of networks trained with different values of l can be seen in Figure 8. Small values of l mean that loss and backpropagation are performed for short physical timespans during training, i.e. a small temporal lookahead, which produces a controller with a deteriorated performance. In addition, performance does not change significantly beyond $l = 16$. Since the same number of iterations were conducted for all runs, using larger time horizons only increases training time. Therefore,

Table 5: Parameters of network trained with differentiable physics.

Hyperparameter	2 DOF	3 DOF
β_{xy}	15	5
$\beta_{\dot{x}}$	5	5
β_F	0.1	0.1
$\beta_{\Delta F}$	2	1
β_{prox}	0.1	0.1
β_α	-	30
$\beta_{\dot{\alpha}}$	-	0.05
$\beta_{\Delta T}$	-	1
n_i	1000	5000
l	16	16

we choose $l = 16$ since it provides a good balance between performance and training time.
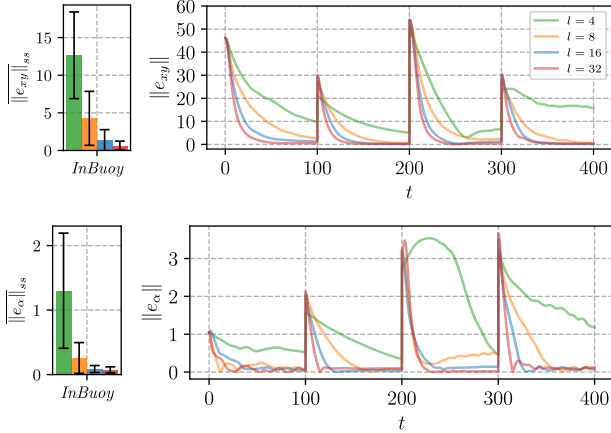
Figure 8: Steady state errors (left) and errors norms (right) for a test with *InBuoy* parameters. Training with small time horizons l produces poor performant controllers. When using a very large time horizon, such as $l = 32$, the gains are not worth the increased training times.

Sensitivity to Weights Initialization In order to evaluate how the performance from the network trained with differentiable physics is influenced by the networks weights initial values, we perform three training runs with different initial seeds. We run a test simulation with parameters from *InBuoy* and objectives that describe a N shape trajectory. Despite minor differences the error norms for all seeds have a similar tracking performance. This is especially apparent when comparing it to other controllers, as shown in Figure 9.

Supervised Learning In total $n_i = 150,000$ iterations are performed. A learning rate of 0.01 is used with learning rate decay so that every 15,000 iterations it drops to half of its size. Training with more data is examined for assessing if the performance of the controller acquired via supervised
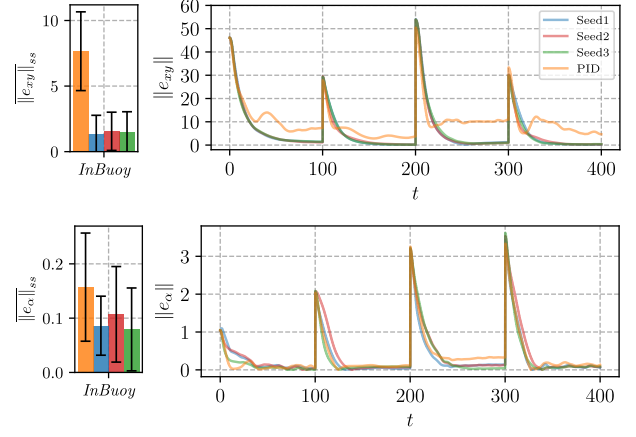

Figure 9: Steady state errors (left) and errors norms (right) for a test with *InBuoy* parameters. The networks have the same parameters but different initialization seeds. Similar tracking performance is achieved for all of them, with Seed 2 having a slightly worse angular tracking.

learning could be improved. A dataset with 200 simulations (double the size of the original one) is generated with 180 simulations being used for training and 20 for validation. Then a test simulation with parameters from *InBuoy* and objectives describing a N shaped trajectory is conducted. It can be seen that adding more data did not result in clear improvements, as shown in Figure 10.
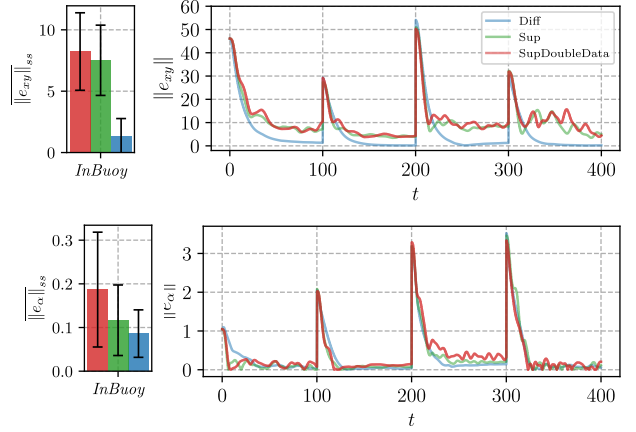

Figure 10: Steady state errors (left) and errors norms (right) for a test with *InBuoy* parameters. A larger dataset does not improve model performance.

Reinforcement Learning We use the SAC implementation from stable-baselines3 (Raffin et al. (2021)). A fixed learning rate of 0.0003 and a reward discount factor γ of 0.99 were chosen. The parameter τ controlling the Polyak Averaging of the two Q-Functions within the critic is set to 0.05. The training takes place until $n_i = 500,000$ iterations are performed.

Hardware and Software All optimization procedures were conducted utilizing the PyTorch framework (Paszke et al. (2019)) on a GeForce RTX 2080 Ti. Approximate training times are displayed in Table 6.

Table 6: Training times.

Algorithm	2 DOF	3 DOF
Reinforcement	21h	-
Supervised	0.5h	1h
Diff. Physics	0.6h	4h

Directly Optimizing Control Efforts Since we have access to a differentiable solver, we investigated directly optimizing the control efforts to assess if this could be a way of acquiring "ground truth" data. This works by running an entire simulation, updating the control efforts based on the gradients, restart the simulation with the modified control efforts and repeat until convergence. We utilized the same loss function used for training the network with differentiable physics but without the E term for better convergence and also with tuned hyperparameters. An objective 4 spatial units away from the initial position was chosen and simulations with 2 DOF and different amount of time steps were performed.

Optimizations performed in simulations with more time steps in total result into different (slower) error decays, as it can be seen in Figure 11, although the loss function for all of them is the same. Therefore it can not be stated that these are the most optimal solutions, especially when dealing with larger unrolls. We believe the optimizations are getting stuck in a local minima, a common problem when directly optimizing long trajectories.

It is worth pointing that a considerable amount of effort needs to be put in tuning the hyperparameters, especially learning rate. We were not able to achieve convergence for simulations with 1000 time steps likely due to the long computation graph in conjunction with nonlinear operations. Tuning hyperparameters in this setting is extremely cumbersome since it takes 10 hours on the current hardware for each optimization run. Due to likelihood of local minima and convergence problems, we opted for not directly using optimized control efforts as ground truth data.

A.5 Additional Results

In the following we present a collection of additional results that were not shown in the main body of the paper displayed on Table 7 and 8 as well as Figure 12, 13, 14, 15, 16, 17, 18 and 19. For all simulations, the cylinder has $m = 11.78$ and radius $\hat{r} = 5$ while the box has $m = 36$, $I = 4000$, width $w = 20$ and height $h = 6$.
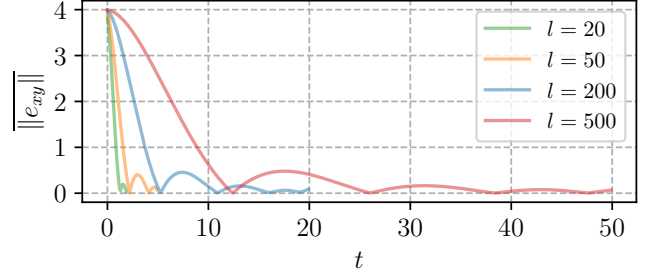

Figure 11: Error norms of simulations with different amount of time steps directly optimized control efforts. Although all runs have the same loss, different decays are obtained.
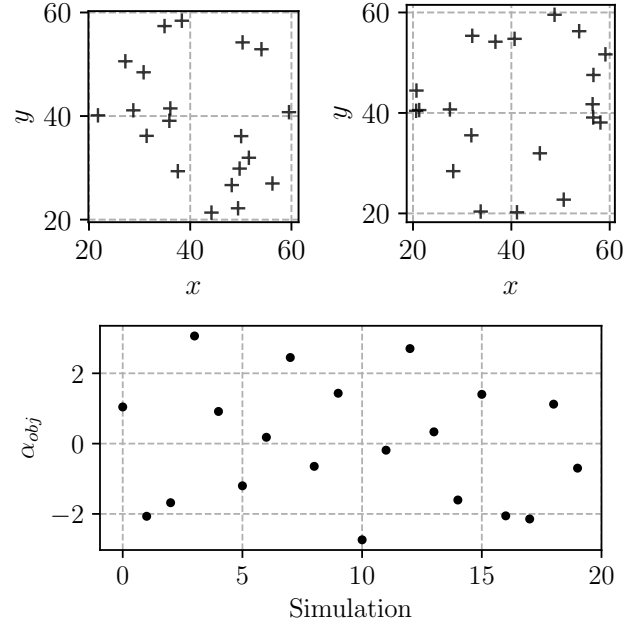

Figure 12: Targets of 2 DOF validation (top-left) and 3 DOF validation (top-right) tests as well as angle targets of 3 DOF validation test (bottom) . The rigid body initial position is located at $(x, y) = (40, 40)$.

Table 7: Spatial steady state error and standard deviation of the considered tests.

Test	RL	LS	Sup	PID	Diff
Validation 2 DOF	$0.6673 \pm \mathbf{0.0007}$	0.5797 ± 0.7233	$\mathbf{0.0563} \pm 0.0203$	0.1386 ± 0.0227	0.08247 ± 0.0924
Validation 3 DOF	-	-	$\mathbf{0.2115} \pm 0.2138$	$0.2206 \pm \mathbf{0.2124}$	0.7762 ± 0.6004
Test with *BuoyNR*	0.7634 ± 0.2587	9.6079 ± 2.1461	2.9695 ± 0.6410	2.2122 ± 0.4086	$\mathbf{0.1943} \pm \mathbf{0.1800}$
Test with *Inflow*	-	-	$3.6632 \pm \mathbf{0.6446}$	3.7965 ± 0.7141	$\mathbf{1.4081} \pm 1.3670$
Test with *InBuoy*	-	-	$7.1705 \pm \mathbf{0.9169}$	8.0952 ± 1.2625	$\mathbf{1.5611} \pm 1.3261$
Test with *Hold*	-	-	16.2431 ± 7.1467	15.0201 ± 6.2401	$\mathbf{4.4825} \pm \mathbf{5.7908}$

Table 8: Angular steady state error and standard deviation of tests.

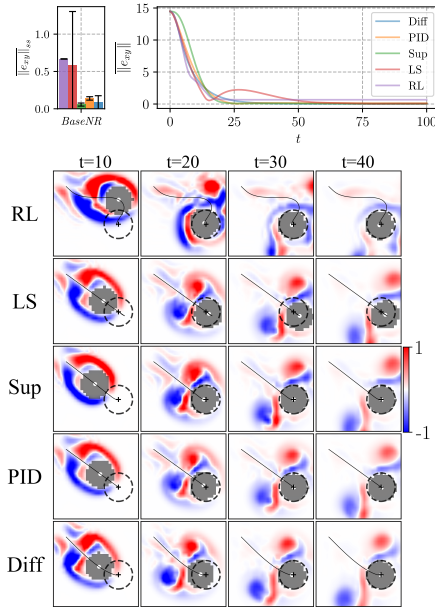| Test	Sup	PID	Diff
Validation 3 DOF	$\mathbf{0.0142} \pm \mathbf{0.0218}$	0.0163 ± 0.0271	0.0183 ± 0.0239
Test with *Inflow*	0.0545 ± 0.0218	$0.0596 \pm \mathbf{0.0187}$	$\mathbf{0.0399} \pm 0.0369$
Test with *InBuoy*	$0.1198 \pm \mathbf{0.0318}$	0.1662 ± 0.0343	$\mathbf{0.0775} \pm 0.0348$
Test with *Hold*	$\mathbf{0.1430} \pm \mathbf{0.0962}$	0.1592 ± 0.0999	0.2269 ± 0.2302

Figure 13: Average spatial error norm of validation simulations with *BaseNR* parameters (top-left) and average steady state error (top-right). All controllers achieve spatial steady state errors smaller than one. Vorticity contours of one simulation from 2 DOF validation test (bottom).
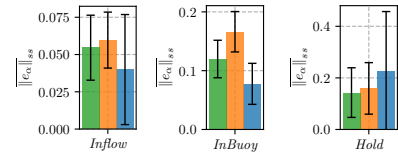

Figure 14: Average angular steady state errors of tests performed with *Inflow* , *InBuoy* and *Hold* parameters (from left to right, respectively).
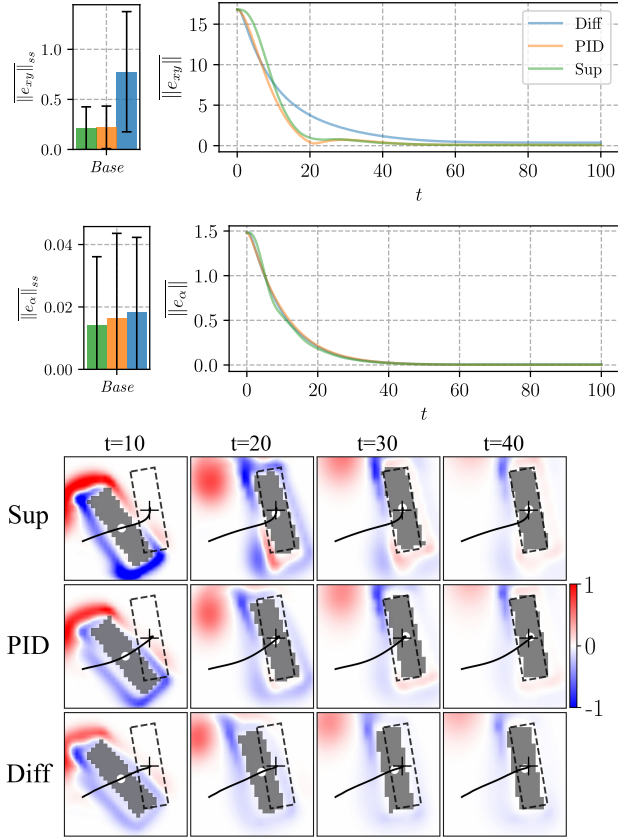
Figure 15: Average steady state errors (top-left, middle-left), trajectories of one of the simulations (bottom) and their error norms (top-right, middle-right) with parameters *Base* . All controllers are able to achieve low steady state errors. The increased spatial steady state error from $\mathcal{P}_{\text{diff}}$ is caused by its slower spatial tracking.
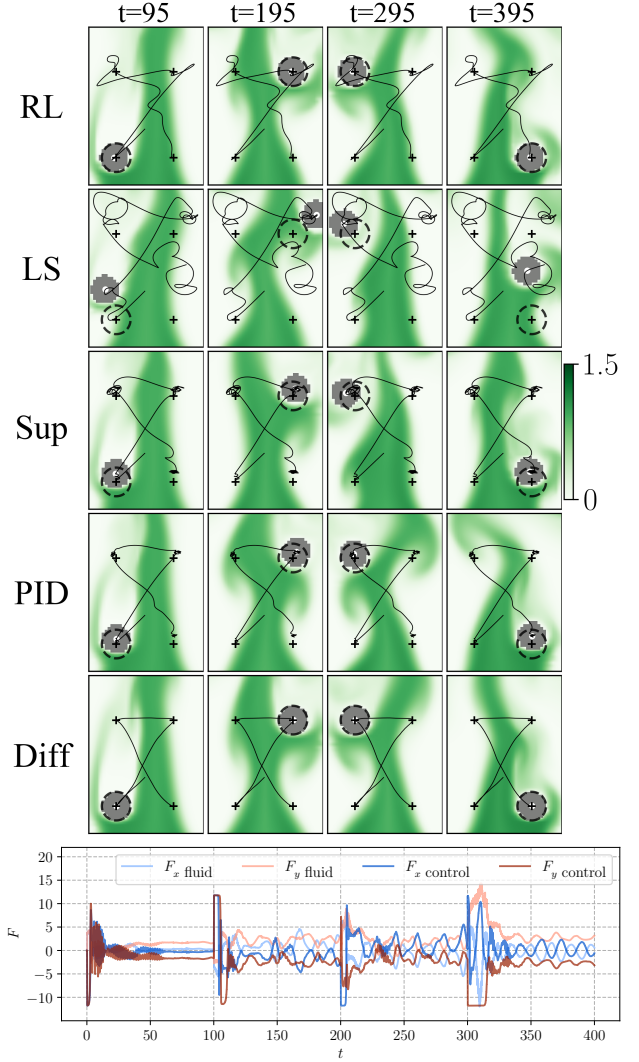

Figure 16: Density contours of lighter fluid (top) of one simulation from test with *BuoyNR* . Fluid and control forces (bottom) from $\mathcal{P}_{\text{diff}}$ run.
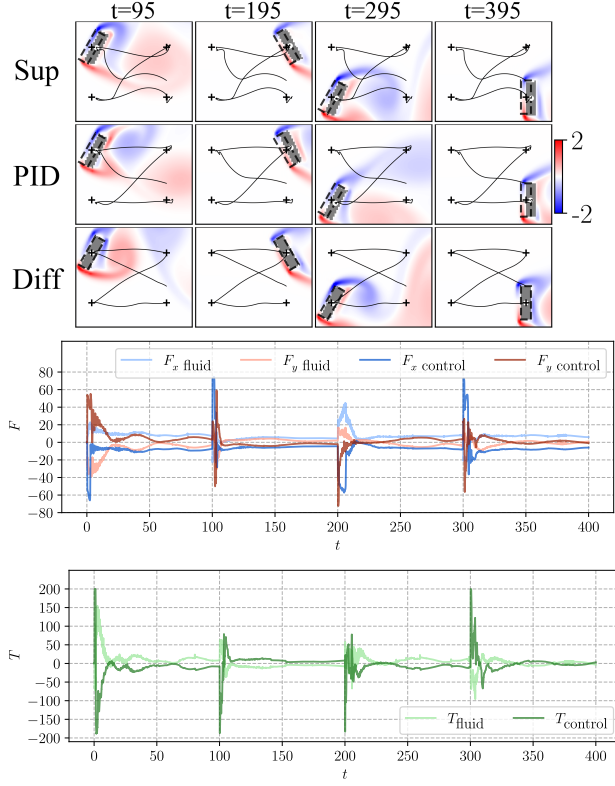
Figure 17: Vorticity contours (top) of one simulation from test with *Inflow* . Fluid and control forces (middle) and torques (bottom) from $\mathcal{P}_{\text{diff}}$ run.
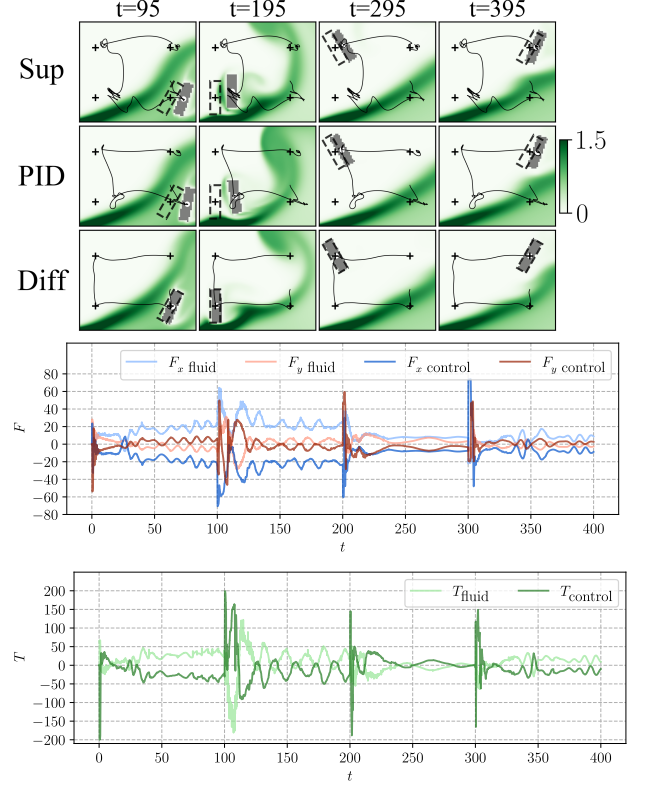

Figure 18: Density contours of lighter fluid (top) of one simulation from test with *InBuoy* . Fluid and control forces (middle) and torques (bottom) from $\mathcal{P}_{\text{diff}}$ run.
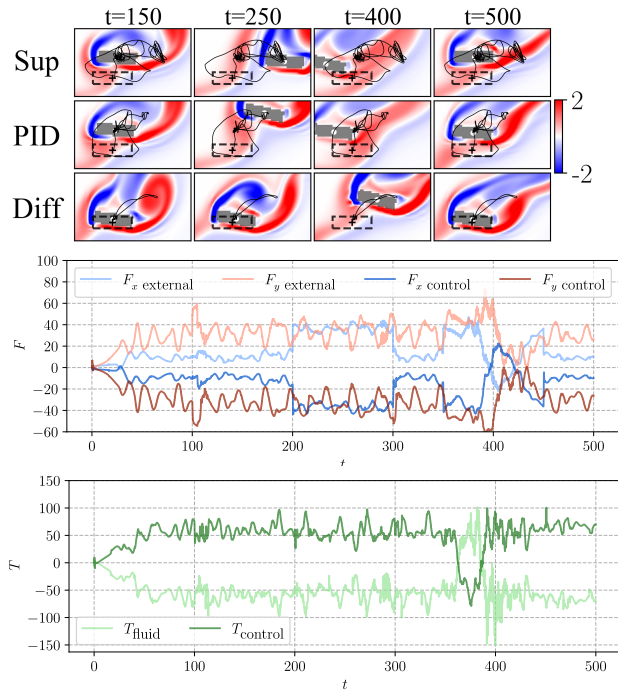
Figure 19: Vorticity contours (top) from test with *Hold* . External (fluid + forcing) and control forces (middle) and torques (bottom) from $\mathcal{P}_{\text{diff}}$ run.