Control of Two-way Coupled Fluid Systems with Differentiable Solvers

Brener L. O. Ramos, Felix Trost & Nils Thuerey
Department of Informatics
Technical University of Munich
Boltzmannstraße 3, 85748 Garching bei München, Germany
brener.ramos@tum.de, ga94zux@mytum.de, nils.thuerey@tum.de

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 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. Further, we show that controllers trained with our approach outperform a variety of classical and learned alternatives in terms of evaluation metrics and generalizing capabilities.

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 & Kuik, 2007; Ho et al., 2003; Lord et al., 2000). Traditionally, open and closed loop control techniques were 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 typically complex and chaotic. Moreover, the number of degrees of freedom in numerical solvers often is too large, making heavily reduced representations necessary (Sipp & Schmid, 2016; Noack et al., 2004; Bergmann & Cordier, 2008; Proctor et al., 2016). Instead, training with a differentiable 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 makes it possible to learn controllers without providing ground truth control signals.

An ubiquitous challenge for neural network approaches is generalization to conditions beyond the training distribution (Goodfellow et al., 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. 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 & 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 et al., 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 & 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 et al., 2019; Li et al., 2020).

Another way of using deep learning for control purposes is through reinforcement learning (Verma et al., 2018; Paris et al., 2021; Ren et al., 2021; Novati et al., 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 & Ermon, 2016; Schulman et al., 2017; Haarnoja et al., 2018b).

Recently, differentiable solvers for 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 find solve for approporiate control efforts of a given task. This task could be placing a piece of cloth into a target container (Liang et al., 2019), pouring liquids (Schenck & Fox, 2018), moving a fluid to a specified region (Holl et al., 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 only relies on a sparse set of measurements from the environment and can be evaluated very efficiently.

3 Methodology

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 equation 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 equation 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 equation 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 equation 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. We additionally target rigid objects immersed in the fluid. Their linear and angular movement can be described by equation 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 equation 6 and equation 7 the system has 3 degrees of freedom (DOF). In a few cases below we will omit equation 7, yielding a 2 DOF scenario. When dealing with a rigid body immersed in a fluid, these terms reduce to

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

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

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 $u = \partial x_r / \partial t$ at rigid body cells.

Together, equation 6 and equation 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:

$$\sum F = -\oint_S p\,\vec{n}(s)\;ds + F_{control} \tag{10}$$

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

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

$$\arg\min_{\theta} \|e_{xy}\| + \|e_{\alpha}\| \tag{12}$$

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

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

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.

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 equation 12, where $z(t)$ is a set of discrete state variables, which we denote with z for brevity, and θ denotes the weights of the network. The network acting as a policy receives the current and previous n_p states as input, and has the objective of infering appropriate control efforts for a given learning objective. Each state consists of the spatial error e_{xy}, angular error e_{α}, rigid

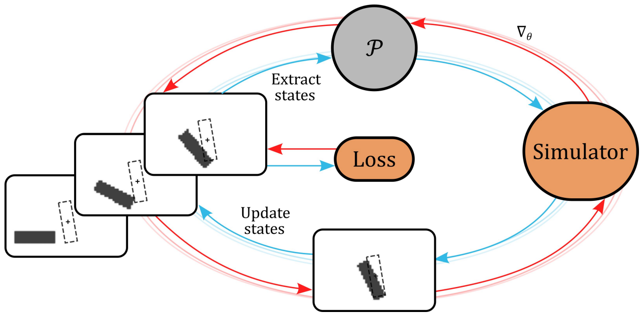


Figure 1: A schematic of the differentiable solver training: Blue arrows represent the forward pass while red ones illustrate the flow of gradients. Importantly, the loss signal is backpropagated through l simulation steps to provide policy $\mathcal{P}$ with long-term feedback about the flow environment.

body linear velocity $\frac{\partial x_r}{\partial t}$, rigid body angular velocity $\frac{\partial \alpha}{\partial t}$ and control efforts $F_{control}$ and $T_{control}$. For clarity, we will refer to the latter simply as F and T. 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 are expressed in the local reference frame of the rigid body. 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 $z = [w^t, w^{t-1}, \ldots, w^{t-n_p}]$, where $w^t = [e^t_{xy}, \dot{x}^t, F^t, e^t_\alpha, \dot{\alpha}^t, T^t]^T$. In the following, a range of different learning procedures are investigated: differentiable physics, supervised and reinforcement learning.

4.1 Learning via a Differentiable Solver

Our method employs a fully differentiable solver. Hence, based on the following loss formulations, we can provide gradients to the neural network policy $\mathcal{P}_{\text{diff}}(z \mid \theta)$ about reactions of the physical system from 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. Our loss formulation include a time horizon of l time steps as a central parameter. 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, allowing it to "plan ahead", e.g., to avoid overshooting. This process is illustrated in figure 1. Instead of pre-computing training data, 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:

$$O = \frac{\beta_{xy}}{l}\sum_{n=0}^{l-1}\|e^n_{xy}\|^2 + \frac{\beta_\alpha}{l}\sum_{n=0}^{l-1}\|e^n_\alpha\|^2, \tag{15}$$

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:

$$V = \frac{\beta_{\dot{x}}}{l}\sum_{n=0}^{l-1}\frac{\|\dot{x}^n\|^2}{\beta_{prox}\|e^n_{xy}\|^2+1} + \frac{\beta_{\dot{\alpha}}}{l}\sum_{n=0}^{l-1}\frac{\|\dot{\alpha}^n\|^2}{\beta_{prox}\|e^n_\alpha\|^2+1} \tag{16}$$

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 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

$$E = \frac{\beta_F}{l}\sum_{n=0}^{l-1}\|F^n\|^2 + \frac{\beta_T}{l}\sum_{n=0}^{l-1}\|T^n\|^2 + \frac{\beta_{\Delta F}}{l}\sum_{n=0}^{l-1}\|F^n - F^{n-1}\|^2 + \frac{\beta_{\Delta T}}{l}\sum_{n=0}^{l-1}\|T^n - T^{n-1}\|^2 \tag{17}$$

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

$$L = O + V + E \tag{18}$$

Above, the β hyperparameters regulate the relative impact of each term for the controller. Because of the direct physical impact of each hyperparameter, adjusting them is straight forward. For example, spatial tracking can be 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. 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 $[\hat{F}(z),\, \hat{T}(z)]^T$. Training with these precomputed forces can be performed in a fully supervised way with the loss

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

4.3 Reinforcement Learning

Additionally, we include a reinforcement learning algorithm that works without making use of the solver gradients. We use the Soft Actor Critic (SAC) algorithm (Haarnoja et al., 2018b), a recent state-of-the-art approach. 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. The actor, which is also represented by a neural network, has the objective of maximizing the predicted Q values. Additionally, SAC introduces an entropy term into the objective of the policy as regularization. It discourages overly confident decisions while also controlling the trade-off between exploration and exploitation inside the action space. In our setting, the actor represents a control policy, and hence we refer to it as $\mathcal{P}_{\text{RL}}(z \mid \theta)$. Following equation 12, the reward at a time t is calculated using the squared distance objective

$$r^t = -\frac{||e^t_{xy}||^2 - ||e^0_{xy}||^2}{||e^0_{xy}||^2} + g(e^t_{xy}), \tag{20}$$

with $g(e^t_{xy})$ yielding a constant G if $||e^t_{xy}|| \leq \epsilon$, and zero otherwise. We normalize r^t by the starting error e^0 from the beginning of a sequence in order to reduce the variance between different trajectories and make the reward independent from the initial state. We present in this paper preliminary results and other reward functions are still being investigated.

5 Experiments

We perform a series of experiments with increasing degrees of complexity to assess the generalization capabilities 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. Source code used in this work can be found on `https://github.com/brenerrr/PhiFlow/tree/two_way_coupling`.

Datasets and Test Scenarios The training data consists of simulations with only one objective and parameters *Ba2* or *Ba3* from Table 1 for systems with 2 and 3 DOF, respectively, using a quiescent flow ($u = 0$) as initial conditions. Since the fluid is initially at rest, all the fluid perturbations are created from the rigid body movement. The validation datasets for *Ba2* and *Ba3* consist of 20 simulations with the same parameters as training but different objectives. To evaluate generalization of 2 DOF networks, we employ *Buo2* , which increases the difficulty by introducing a lighter fluid source that disrupts the flow through buoyancy. For 3 DOF networks trained on *Ba3* , we use the

Table 1: Main parameters of experimental setups. IC denotes initial conditions which are shown in the appendix.

ID	**Domain**	**DOF**	Re	Δt	**Buoyancy**	**Inflow**	**IC**
Ba2	80×80	2	1000	0.1	$\times$	$\times$	A
Buo2	80×80	2	1000	0.05	$\surd$	$\times$	A
Ba3	80×80	3	1000	0.1	$\times$	$\times$	B
In3	175×110	3	3000	0.05	$\times$	$\surd$	C
InBuo3	175×110	3	3000	0.05	$\surd$	$\surd$	C
InBuoAg3	175×110	3	5000	0.05	$\surd$	$\surd$	D

scenarios *In3* , *InBuo3* and *InBuoAg3* . For *In3* , the fluid is less viscous (higher Re) and Δt is smaller. Correspondingly, the controllers are sampled once every two timesteps. Setup *InBuo3* introduces buoyancy forces as well an inflow while a second rotating obstacle is additionally present in *InBuoAg3* . These test environments were designed to deviate more and more strongly from the quiescent training conditions. Unless stated otherwise, the test datasets are comprised of 5 simulations with different trajectories, created by changing the objectives after $\Delta t = 100$. The supervised learning approach uses a separate, pre-computed dataset for training. It is comprised of 100 simulations from which 80 are used for training and 20 for validation and each simulation has 500 time steps.

Neural Network Representation and Training The underlying network architecture for all of them 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 section C.1.3. After 1000 simulation steps we restart the simulation 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. For supervised training, in total $n_i = 150{,}000$ and $n_i = 200{,}000$ training iterations are performed for the network used for systems with 2 and 3 DOF, respectively. The reinforcement learning approach uses the same simulation environment as the differentiable training. After each simulated time step a batch of 128 samples is drawn from the replay buffer and used for training. In total, $n_i = 300{,}000$ training iterations are performed.

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. It is calculated using the last $3/4$ of the time interval in which an objective was being tracked. For bar plots of $\overline{||e||}_{ss}$ we also display one standard deviation for illustrating the spread of the error values.

6 Results

We first perform a broad comparison of the different algorithms, before moving on to more complex cases and generalization tasks.

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 proportional-integrator-derivative (PID) $\mathcal{P}_{\text{PID}}$ and a loop shaping controller $\mathcal{P}_{\text{LS}}$ designed through a blend of mixed-sensitivity design and the Glover-McFarlane method.

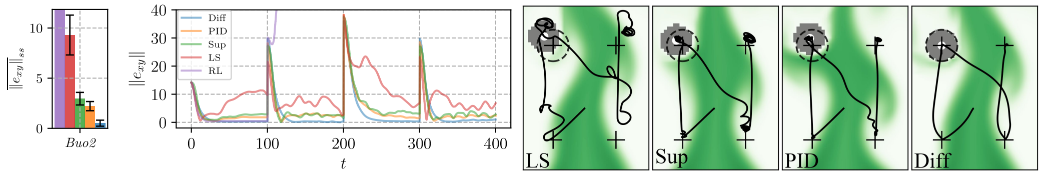


Figure 2: Average spatial steady state errors (left), trajectories of one of the test simulations (right) and their error norms (middle) with parameters *Buo2* . When compared to the other approaches $\mathcal{P}_{\text{diff}}$ exhibit less oscillations and a lower average steady state error.

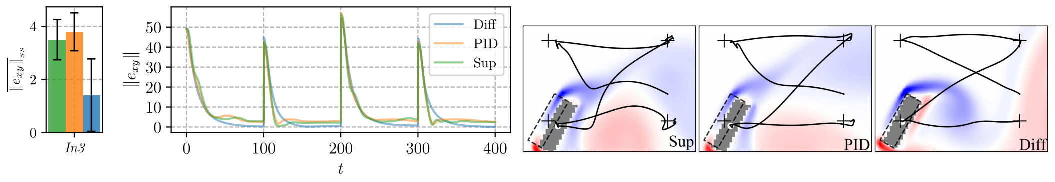


Figure 3: Average steady state errors (left), trajectories of one of the test simulations (right) and their error norms (middle) with parameters *In3* . Trajectories are colored by lighter fluid ($t = 295$). All approaches accomplish a similar angle tracking performance but $\mathcal{P}_{\text{diff}}$ has a better spatial one.

2 DOF Validation First, we consider a validation with the setup *Ba2* (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 can be found in the appendix.

Increased Difficulty Next we perform a test with the setup *Buo2* , which includes perturbations from a lighter fluid and a longer time window. The interaction between buoyancy and rigid body creates oscillatory flow structures and the control task becomes more challenging. As a consequence, $\overline{\|e_{xy}\|}_{ss}$ is much larger for all approaches except for $\mathcal{P}_{\text{diff}}$. The latter exhibits an error more than three times smaller than the second best (PID), as shown in figure 2. The trajectories clearly show the amount of undesirable oscillations and overshooting present in the $\mathcal{P}_{\text{LS}}$, $\mathcal{P}_{\text{sup}}$ and $\mathcal{P}_{\text{PID}}$. Both $\mathcal{P}_{\text{LS}}$ and $\mathcal{P}_{\text{RL}}$ exhibit a poor performance, with the latter still being preliminary results and not able to finish all test simulations. Hence, we will omit these two methods for further tests.

6.2 Increased Complexity and Generalization

We now evaluate the performance of the remaining methods for a rectangular body with orientation (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, i.e. from *Ba3* . All three remaining methods fare well for these environments, and yield stable controllers (details are provided in the appendix).

Generalization Tests *Modified Inflow Conditions.* Next, a test with the *In3* setup is 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. Next, we introduce a source of lighter fluid that rises due to buoyancy at the bottom of the domain with parameters *InBuo3* . This test has the goal of assessing the robustness

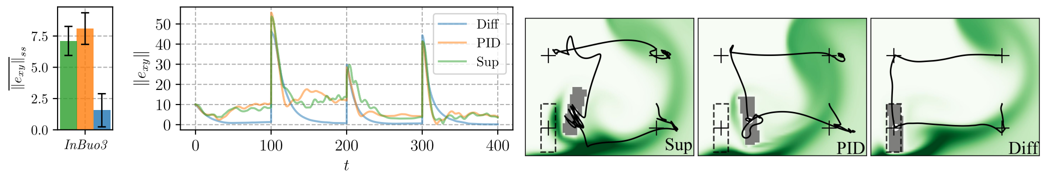


Figure 4: Average steady state errors (left), trajectories of one of the test simulations (right) and their error norms (middle) with parameters *InBuo3* . Trajectories are colored by lighter fluid ($t = 195$). Buoyancy worsens the performance of controllers but $\mathcal{P}_{\text{diff}}$ still maintains low steady state errors.

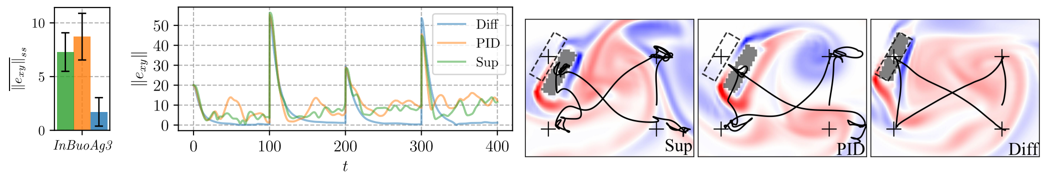


Figure 5: Average steady state errors (left), trajectories of one of the test simulations (right) and their error norms (middle) with parameters *InBuoAg3* . Trajectories are colored by vorticity ($t = 295$). Although the flow is fairly chaotic, $\mathcal{P}_{\text{diff}}$ is able to maintain low steady state errors and good tracking.

of the controllers further, since the fluid source tends to create higher frequency oscillations, which makes the control task harder. While the $\mathcal{P}_{\text{sup}}$ and $\mathcal{P}_{\text{PID}}$ have a considerable worsening in their performance with higher steady state errors, they also display undesirable oscillations. On the other hand, $\mathcal{P}_{\text{diff}}$ maintains low values for $\overline{\|e_{xy}\|}_{ss}$ and $\overline{\|e_{\alpha}\|}_{ss}$ while rejecting most of the perturbations as shown in figure 4.

Agitated Inflow with Buoyancy. The last test introduces a second rigid body which spins close to the left boundary, together with a higher $Re = 5000$. These characteristics contribute to a more chaotic flow, with stronger high frequency perturbations resulting in a very challenging test scenario. Especially when considering the spatial error, it is clear that $\mathcal{P}_{\text{diff}}$ is able to stabilize most perturbations and maintain a low steady state error as shown in figure 5. The approach can maintain a stable path despite the changed flow conditions, while the other approaches fail to do so.

6.3 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 when training took place in a different, simplified environment. $\mathcal{P}_{\text{diff}}$ is able to handle new situations without the strongly deteriorated tracking performance of the other controllers, with our tests indicating 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 appealing to be able to use the original set of equations of a system, even if they are as nonlinear as the NS equations because reduced representations often do not portrait well nonlinear behaviors. Instead, the differentiable simulation approach allows training the controller using the full set of model equations, such that it can adapt to the subtleties of the environment.

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. 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.

ACKNOWLEDGMENTS

This work was supported by the ERC Consolidator Grant *SpaTe* (CoG-2019-863850).

REFERENCES

Thanasis Barlas and Gijs Kuik. State of the art and prospectives of smart rotor control for wind turbines. *Journal of Physics: Conference Series*, 75:012080, 08 2007. doi: 10.1088/1742-6596/75/1/012080.

M. Bergmann and L. Cordier. 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, 2008. ISSN 0021-9991.

Katharina Bieker, Sebastian Peitz, Steven L. Brunton, J. Nathan Kutz, and Michael Dellnitz. Deep model predictive flow control with limited sensor data and online learning. *Theoretical and Computational Fluid Dynamics*, 34(4):577–591, 2020.

Kaixuan Chen, Jin Lin, Yiwei Qiu, Feng Liu, and Yonghua Song. Deep learning-aided model predictive control of wind farms for agc considering the dynamic wake effect. *Control Engineering Practice*, 116:104925, 2021.

Scott Samuel Collis, Ronald D. Joslin, Avi Seifert, and Vassilis Theofilis. Issues in active flow control: theory, control, simulation, and experiment. *Progress in Aerospace Sciences*, 40(4):237–289, 2004. ISSN 0376-0421.

Hamidreza Eivazi, Hadi Veisi, Mohammad Hossein Naderi, and Vahid Esfahanian. Deep neural networks for nonlinear model order reduction of unsteady flows. *Physics of Fluids*, 32(10):105104, 2020.

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

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

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

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

Steven Ho, Hany Nassef, Nick Pornsinsirirak, Yu-Chong Tai, and Chih-Ming Ho. Unsteady aerodynamics and flow control for flapping wing flyers. *Progress in Aerospace Sciences*, 39(8):635–681, 2003. ISSN 0376-0421.

Philipp Holl, Nils Thuerey, and Vladlen Koltun. Learning to control pdes with differentiable physics. In *International Conference on Learning Representations*, 2020.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

John Ingraham, Adam Riesselman, Chris Sander, and Debora Marks. Learning protein structure with a differentiable simulator. In *International Conference on Learning Representations*, 2019.

Huibert Kwakernaak. Mixed sensitivity design. *IFAC Proceedings Volumes*, 35(1):61–66, 2002. ISSN 1474-6670. 15th IFAC World Congress.

Yunzhu Li, Hao He, Jiajun Wu, Dina Katabi, and Antonio Torralba. Learning compositional koopman operators for model-based control. In *International Conference on Learning Representations*, 2020.

Junbang Liang, Ming C. Lin, and Vladlen Koltun. *Differentiable Cloth Simulation for Inverse Problems*. Curran Associates Inc., 2019.

Wesley Lord, Douglas MacMartin, and Gregory Tillman. *Flow control opportunities in gas turbine engines*. 2000.

Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(4): 1–11, 2018.

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

Jeremy Morton, Antony Jameson, Mykel J Kochenderfer, and Freddie Witherden. Deep dynamical modeling and control of unsteady fluid flows. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

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

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

Guido Novati, Lakshminarayanan Mahadevan, and Petros Koumoutsakos. Controlled gliding and perching through deep-reinforcement-learning. *Physical Review Fluids*, 4(9):093902, 2019.

Romain Paris, Samir Beneddine, and Julien Dandois. Robust flow control and optimal sensor placement using deep reinforcement learning. *Journal of Fluid Mechanics*, 913:A25, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Joshua L. Proctor, Steven L. Brunton, and J. Nathan Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

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

C. Schenck and D. Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Proceedings of the Second Conference on Robot Learning (CoRL)*, Zurich, Switzerland, 2018.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Denis Sipp and Peter J. Schmid. Linear Closed-Loop Control of Fluid Instabilities and Noise-Induced Perturbations: A Review of Approaches and Tools1. *Applied Mechanics Reviews*, 68(2), 05 2016. ISSN 0003-6900. 020801.

Tetsuya Takahashi, Junbang Liang, Yi-Ling Qiao, and Ming C. Lin. Differentiable fluids with solid coupling for learning and control. In *AAAI*, 2021.

Marc Toussaint, Kelsey R. Allen, Kevin A. Smith, and Joshua B. Tenenbaum. 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*, pp. 6231–6235. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

Siddhartha Verma, Guido Novati, and Petros Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23):5849–5854, 2018. ISSN 0027-8424.

Enoch Yeung, Soumya Kundu, and Nathan Hodas. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. In *2019 American Control Conference (ACC)*, pp. 4832–4839, 2019.

A LINEAR CONTROLLERS

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

$$U^t = Pe^t + D\frac{e^t - e^{i-1}}{0.1} + 0.1\sum_{n=0}^{i} e^n I \tag{21}$$

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. It should be 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

$$U^t = \sum_{p=0}^{2} e^{i-p} n_p - \sum_{p=1}^{2} U^{i-p} d_p \tag{22}$$

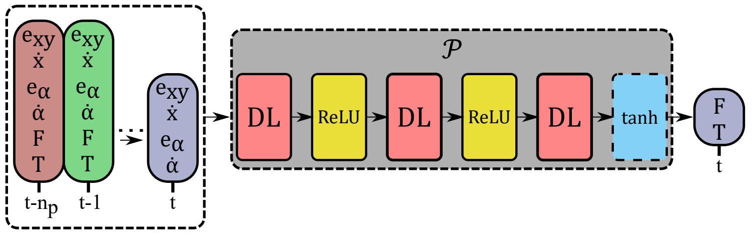


Figure 6: Visualization of the neural network architecture.

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	$-1.4694211940919182\text{e}00$	$-1.2306775904257603\text{e}00$
2	$3.0598060140064326\text{e}{-01}$	$2.6726488821832250\text{e}{-01}$

B NETWORKS ARCHITECTURE DETAILS

The dimensions of the considered neural network layers are given in Table 4, and its architecture is visualized in figure 6. 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]	[33, 32]
1	[38, 38]	[32, 32]
2	[38, 2]	[32, 3]

C TRAINING DETAILS

C.1 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 100 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.

C.1.1 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 *InBuoAg3* and a *N* shaped trajectory is conducted and the performance of networks trained with different values of l can be seen in figure 7. 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, we choose $l = 16$ since it provides a good balance between performance and training time.

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
$\beta_{\Delta F}$	0	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

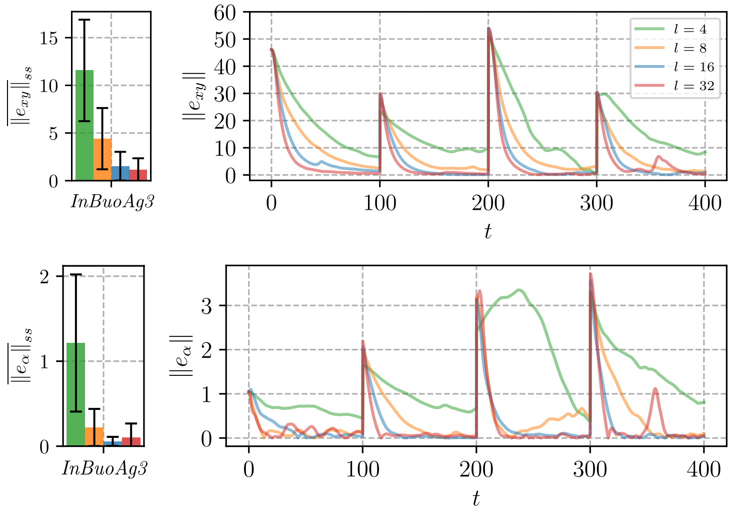


Figure 7: Steady state errors (left) and errors norms (right) for a test with *InBuoAg3* parameters. Training with small time horizons l produces poor performant controllers. When using a very large time horizon, such as $l = 32$, the gains are not worth the doubled amount of training time.

C.1.2 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 *InBuoAg3* 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 8.

C.1.3 Ablation Studies

We present an evaluation study with *InBuo3* to assess the influence of 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 9. 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

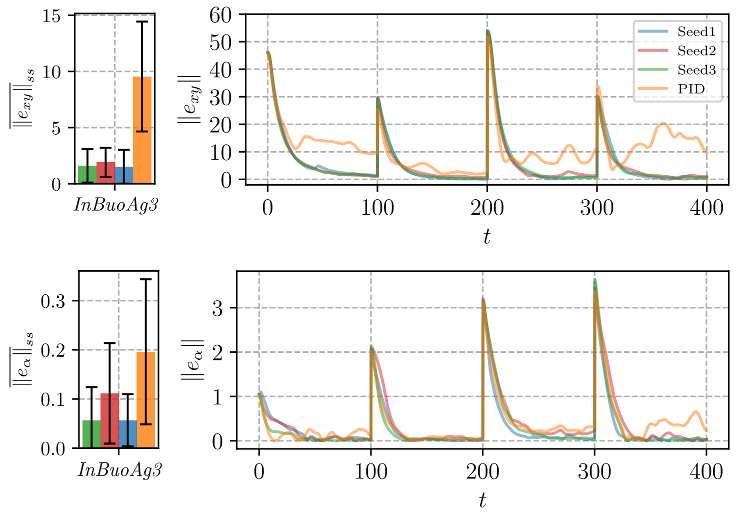


Figure 8: Steady state errors (left) and errors norms (right) for a test with *InBuoAg3* parameters. The networks have the same parameters but different initialization seeds. Similar tracking performance is achieved for all of them, with Seed 2 having a slightly worse angular tracking.

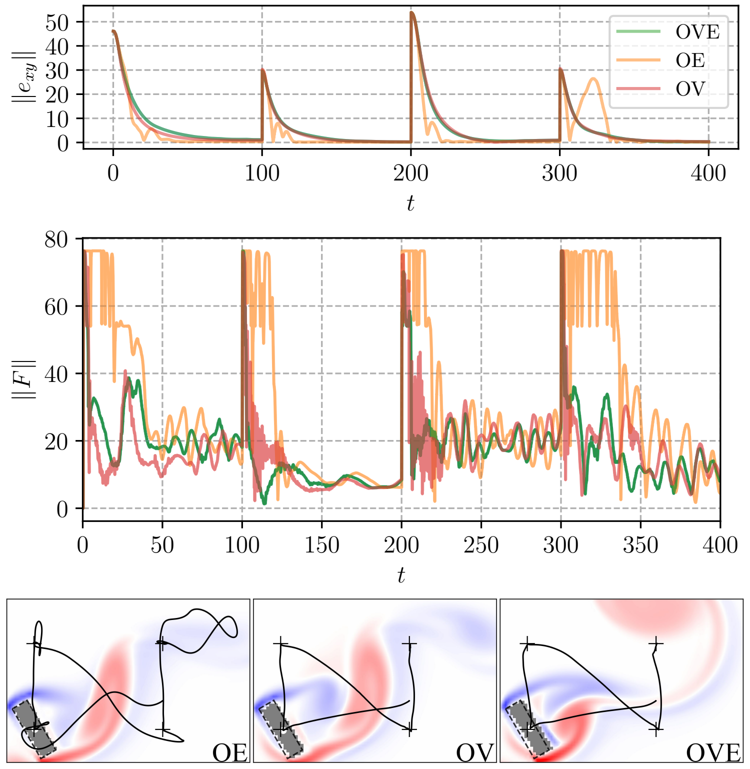


Figure 9: From top to bottom: errors norm, control efforts norm and trajectories of controllers colored by vorticity at $t = 95$ on test with *InBuo3* parameters (bottom). All loss terms are necessary in order to accomplish steady error decay and maintain small and smooth control efforts.

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.

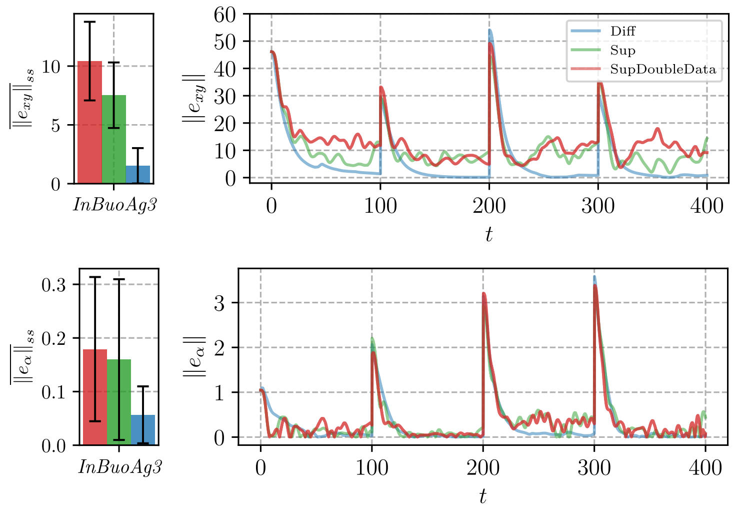


Figure 10: Steady state errors (left) and errors norms (right) for a test with InBuoAg3 parameters. A bigger dataset does not improve model performance.

C.2 Supervised Learning

A learning rate of 0.01 is used with learning rate decay. 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 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 *InBuoAg3* 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.

C.3 Reinforcement Learning

We use the SAC implementation from stable-baselines3 (Raffin et al., 2021). We use a fixed learning rate of 0.0003 and a reward discount factor γ of 0.99. The parameter τ controlling the Polyak Averaging of the two Q-Functions within the critic is set to 0.05.

C.4 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	17h	-
Supervised	0.5h	1h
Diff. Physics	0.6h	4h

D Additional Results

In the following we present a collection of additional results that were not shown in the main body of the paper. 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 $h = 6$.

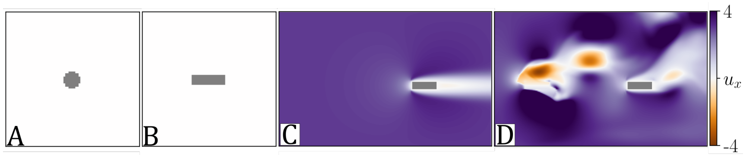


Figure 11: x component of velocity field of initial conditions.

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

Test	**RL**	**LS**	**Sup**	**PID**	**Diff**
Validation 2 DOF	0.5047 ± 0.1065	0.5797 ± 0.7233	$\mathbf{0.0563} \pm \mathbf{0.0203}$	0.1386 ± 0.0227	0.1363 ± 0.1257
Validation 3 DOF	-	-	$\mathbf{0.1540} \pm \mathbf{0.2053}$	0.2206 ± 0.2124	0.7762 ± 0.6004
Test with *Buo2*	-	9.2907 ± 1.9833	2.9562 ± 0.6263	2.2148 ± 0.4577	$\mathbf{0.5153} \pm \mathbf{0.3002}$
Test with *In3*	-	-	3.5010 ± 0.7599	$3.7965 \pm \mathbf{0.7141}$	$\mathbf{1.4081} \pm 1.3670$
Test with *InBuo3*	-	-	$7.1072 \pm \mathbf{1.1612}$	8.0952 ± 1.2625	$\mathbf{1.5611} \pm 1.3261$
Test with *InBuoAg3*	-	-	7.2815 ± 1.7926	8.7141 ± 2.1694	$\mathbf{1.7167} \pm \mathbf{1.3238}$

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

Test	**Sup**	**PID**	**Diff**
Validation 3 DOF	$\mathbf{0.0149} \pm 0.0241$	0.0163 ± 0.0271	0.0183 ± 0.0239
Test with *In3*	0.0553 ± 0.0184	0.0596 ± 0.0187	$\mathbf{0.0399} \pm \mathbf{0.0369}$
Test with *InBuo3*	0.1593 ± 0.0452	$0.1662 \pm \mathbf{0.0343}$	$\mathbf{0.0775} \pm 0.0348$
Test with *InBuoAg3*	0.1643 ± 0.0602	0.1564 ± 0.0607	$\mathbf{0.0702} \pm \mathbf{0.0399}$

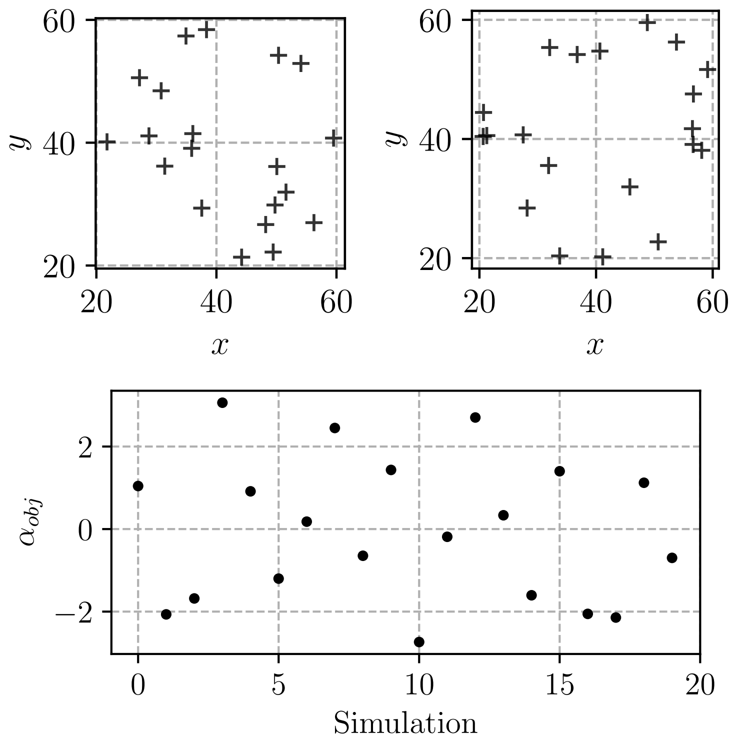


Figure 12: Targets of 2 DOF validation (top-left) and 3 DOF validation (top-right) tests as well as angle targets of 3 DOF validation test (bottom) . The rigid body initial position is located at $(x, y) = (40, 40)$.

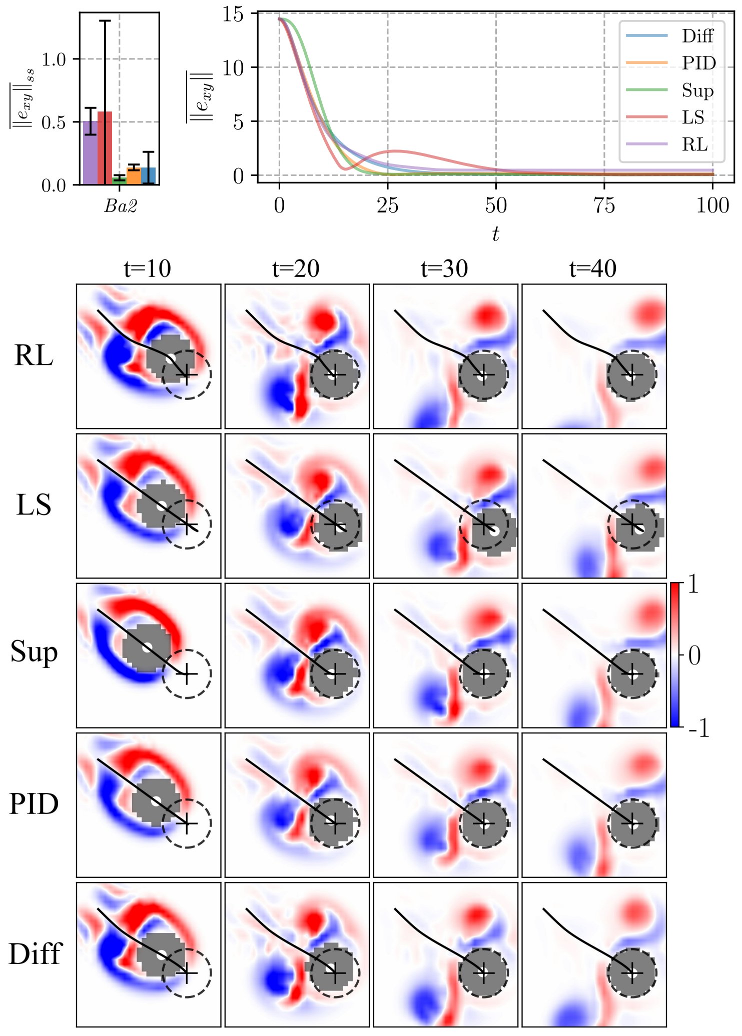


Figure 13: Average spatial error norm with *Ba2* parameters (top-left) and average steady state error (top-right). All controllers achieve spatial steady state errors smaller than one. Vorticity contours of one simulation from 2 DOF validation test (bottom).

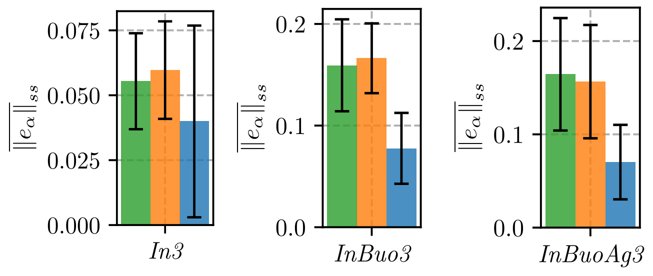


Figure 14: Average angular steady state errors of tests performed with *In3* , *InBuo3* and *InBuoAg3* parameters (from left to right, respectively).

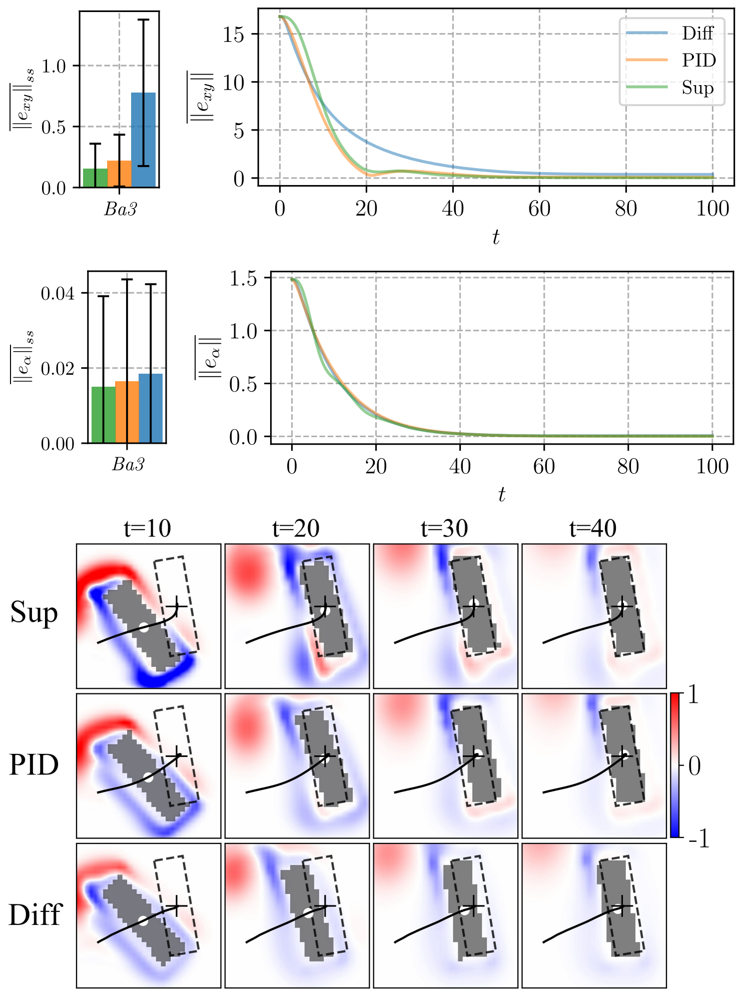


Figure 15: Average steady state errors (top-left, middle-left), trajectories of one of the simulations (bottom) and their error norms (top-right, middle-right) with parameters *Ba3* . All controllers are able to achieve low steady state errors. The increased spatial steady state error from $\mathcal{P}_{\text{diff}}$ is caused by its slower spatial tracking.

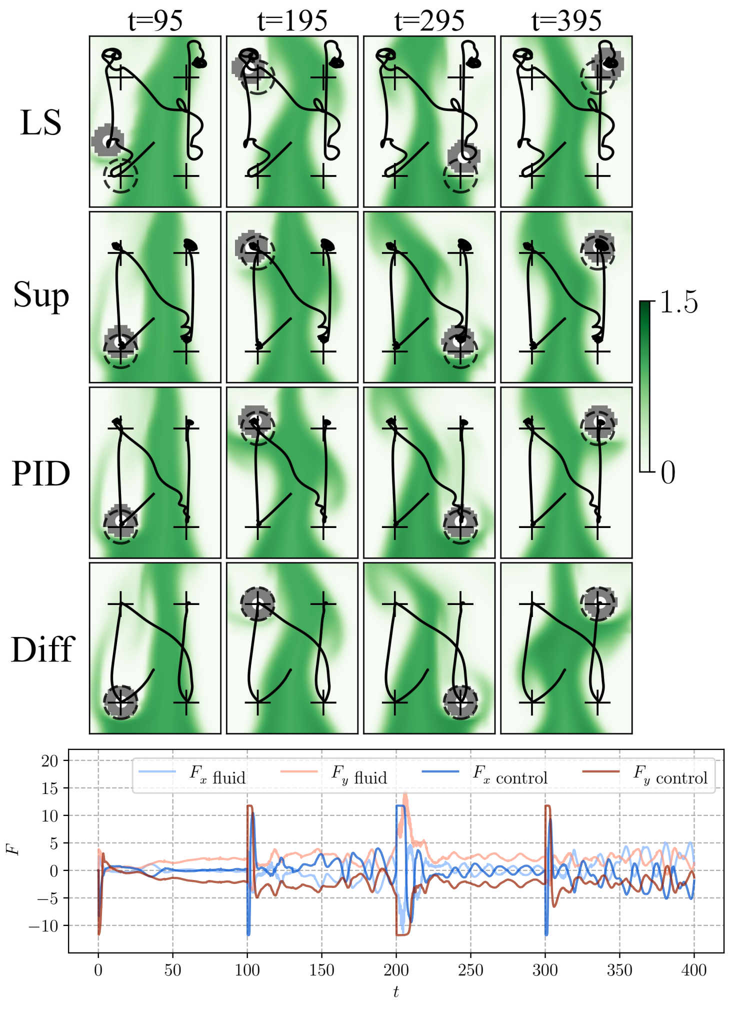


Figure 16: Density contours of lighter fluid (top) of one simulation from test with *Buo2* . Fluid and control forces (bottom) from $\mathcal{P}_{\text{diff}}$ run.

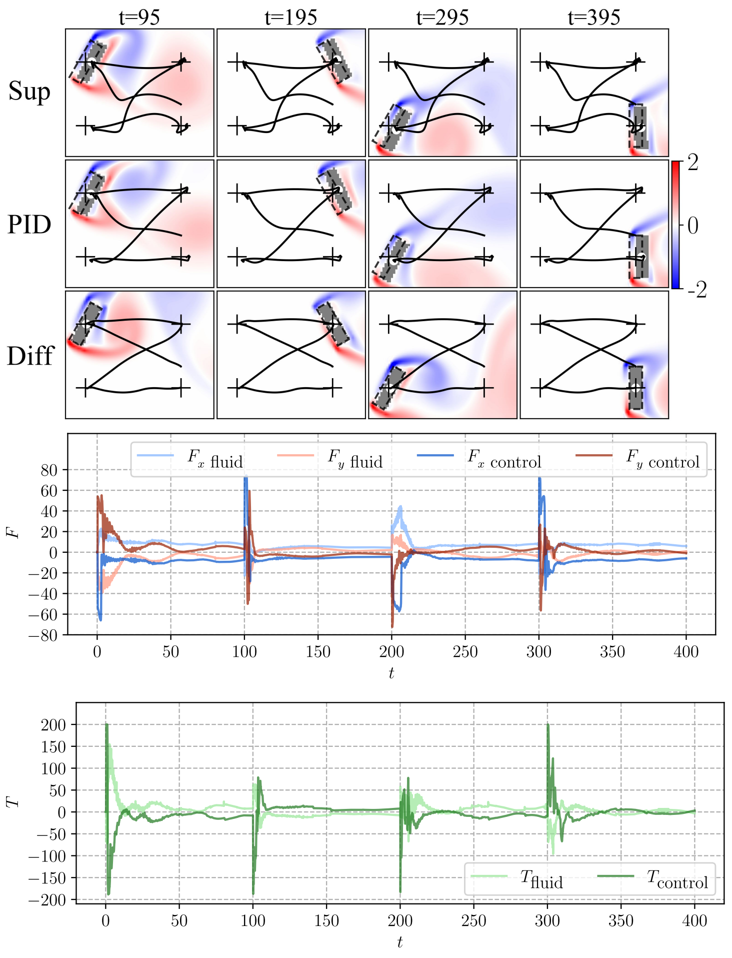
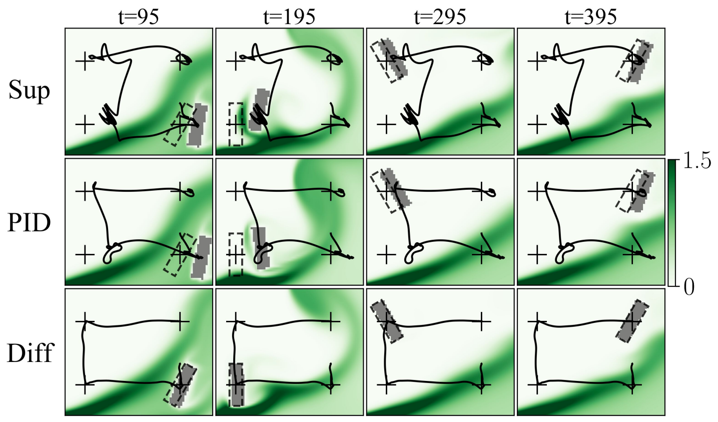
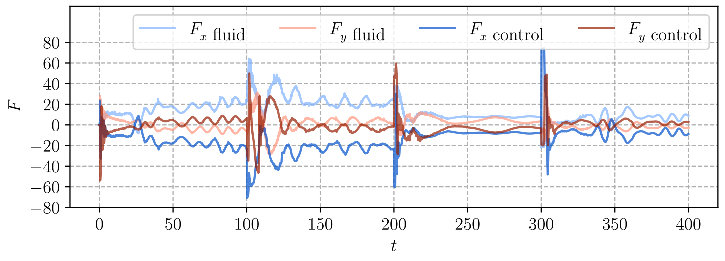


Figure 17: Vorticity contours (top) of one simulation from test with *In3* . Fluid and control forces (middle) and torques (bottom) from $\mathcal{P}_{\text{diff}}$ run.





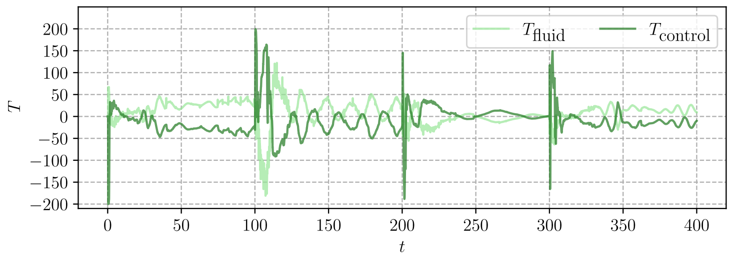


Figure 18: Density contours of lighter fluid (top) of one simulation from test with *InBuo3* . Fluid and control forces (middle) and torques (bottom) from $\mathcal{P}_{\text{diff}}$ run.

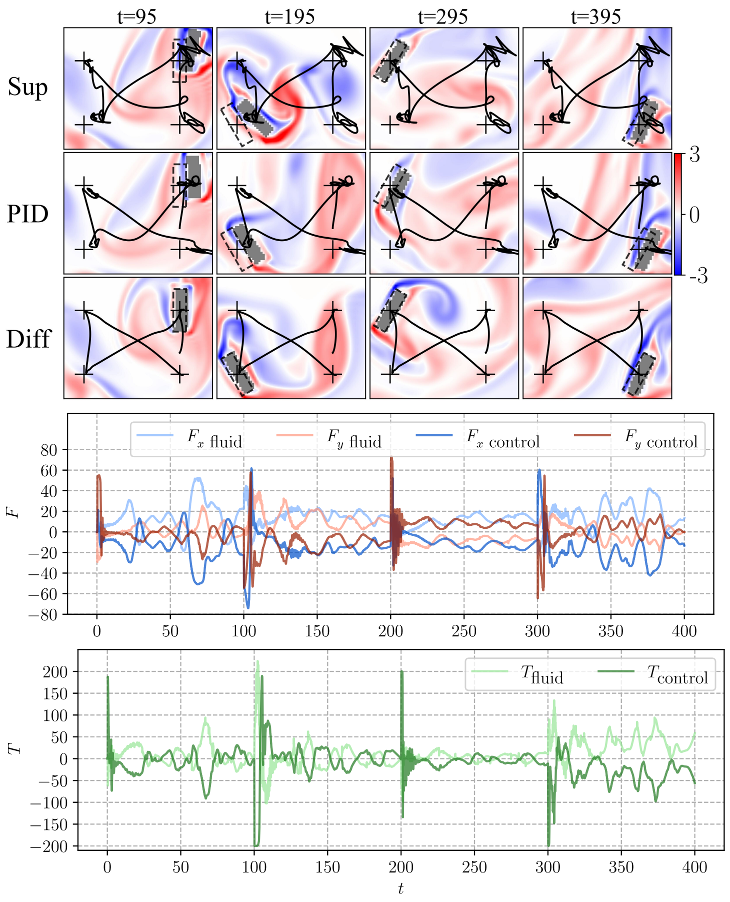


Figure 19: Vorticity contours (top) of one simulation from test with *InBuoAg3* . Fluid and control forces (middle) and torques (bottom) from $\mathcal{P}_{\text{diff}}$ run.