Versatile Control of Fluid-Directed Solid Objects Using Multi-Task Reinforcement Learning

BO REN*, College of Computer Science, Nankai University, China
XIAOHAN YE*, College of Computer Science, Nankai University, China
ZHERONG PAN, Lightspeed & Quantum Studios, Tencent America, USA
TAIYUAN ZHANG, College of Computer Science, Nankai University, China

We propose a learning-based controller for high-dimensional dynamic systems with coupled fluid and solid objects. The dynamic behaviors of such systems can vary across different simulators and the control tasks subject to changing requirements from users. Our controller features high versatility and can adapt to changing dynamic behaviors and multiple tasks without re-training, which is achieved by combining two training strategies. We use meta-reinforcement learning to inform the controller of changing simulation parameters. We further design a novel task representation, which allows the controller to adapt to continually changing tasks via hindsight experience replay. We highlight the robustness and generality of our controller on a row of dynamic-rich tasks including scooping up solid balls from a water pool, in-air ball acrobatics using fluid spouts, and zero-shot transferring to unseen simulators and constitutive models. In all the scenarios, our controller consistently outperforms the plain multi-task reinforcement learning baseline.

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: fluid/solid coupling, optimal control, reinforcement learning

ACM Reference Format:
Bo Ren, Xiaohan Ye, Zherong Pan, and Taiyuan Zhang. 2022. Versatile Control of Fluid-Directed Solid Objects Using Multi-Task Reinforcement Learning. *ACM Trans. Graph.* 1, 1 (July 2022), 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

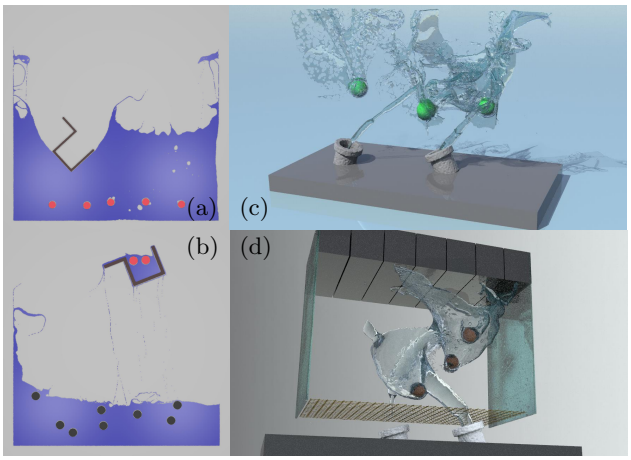


Fig. 1. We highlight our 4 benchmarks of coupled fluid-solid control tasks: (a) Controlling a spoon to scoop up as many balls from the water pool as possible; (b) Controlling a spoon to scoop up all red balls from the water pool, while avoiding picking up the black ones; (c) Controlling the pose and out-flux of two water spouts to keep three balls balanced; (d) Controlling two water spouts to push the balls and hit the music keys with precise timing, according to online-selected music scripts. In all these scenarios, we train the agents in low-res simulators and transfer the learned skills to a high-res simulator online to generate high-quality goal-directed animations.

1 INTRODUCTION

While physics-based fluid animations have achieved an unprecedented visual realism, their dynamic behaviors are notoriously difficult to modify or edit by users. This is largely due to their high-dimensionality and strong coupling between degrees-of-freedom. Over the years, graphic researchers have proposed various fluid editing techniques including ghost forces [McNamara et al. 2004; Pan and Manocha 2017a; Tang et al. 2021; Treuille et al. 2003], shape blending [Raveendran et al. 2014; Thuerey 2016], and stylization [Sato et al. 2018]. However, these methods inconvenience users by either injecting non-physical components into the animation (e.g., ghost control forces), taking strong assumptions on the fluid flow (e.g., single-phase flow, no free surface), or incurring unacceptably high computational costs (typically hours to days).

Technically, the two main obstacles towards ideal fluid controllers are non-smoothness and underactuation. Several recent researches [Hu et al. 2019a,c] highlight the efficacy of modal-based control of (possibly high-dimensional) robots guided by differentiable dynamic systems. However, it is unlikely for these techniques to succeed in fluid control tasks

*Both authors contributed equally to this research.

Authors' addresses: Bo Ren, rb@nankai.edu.cn, College of Computer Science, Nankai University, Tianjin, China; Xiaohan Ye, yexiaohan@mail.nankai.edu.cn, College of Computer Science, Nankai University, Tianjin, China; Zherong Pan, Lightspeed & Quantum Studios, Tencent America, Seattle, USA, zrpan@tencent.com; Taiyuan Zhang, College of Computer Science, Nankai University, Tianjin, China, imaginer.tai@gmail.com.

0730-0301/2022/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

due to the ubiquitous, discontinuous fluid-solid boundary conditions, breaking the differentiable assumption. On the other hand, classical fluid control algorithms [Pan and Manocha 2017a; Tang et al. 2021; Treuille et al. 2003] assume a single-phase flow where ghost forces can be injected everywhere in the simulated domain, i.e., fluid dynamics are fully actuated. This assumption allows the ghost forces to affect the control objective instantaneously, based on which several prior works [Fattal and Lischinski 2004; Raveendran et al. 2012] only need to consider a single timestep of simulation. For higher visual realism, however, it is preferred to keep the fluid dynamic model intact and only change the boundary conditions. Further, many control problems involve two-way fluid-solid coupling where the single-phase assumption does not hold.

Recently, large-scale deep reinforcement learning (DRL) has opened doors to complex real-time decision-making, involving many graphical problems like character motion control, material manipulation, and co-design [Ma et al. 2018; Peng et al. 2018, 2017; Spielberg et al. 2019; Zhang et al. 2020]. DRL has previously been adopted by [Ma et al. 2018] to control two-way coupled liquid-solid systems where control signals are only applied as boundary conditions. However, their controller is optimized to solve a single task for a specific-type of fluid simulator and parameters, which significantly limits their applications in the graphics community. Taking interactive games for example, a controller may be adapted to multiple tasks based on user's control signals. For animation editing applications, users might modify the fluid simulation environment (such as resolution, viscosity, etc.) and the controller should adapt to these changes without re-training. Therefore, solving the general problem of multi-task fluid-directed object control will contribute to more diversities in animation editing phases of the movie industry, and concretely help interactive virtual game designers.

Main Result: We propose a novel policy search algorithm for coupled fluid-solid dynamic systems, where the trained controllers have a high universal performance over a distribution of different simulation environments and tasks. Our method is inspired by the meta-RL techniques [Finn et al. 2017; Gupta et al. 2018; Rakelly et al. 2019]. Specifically, we keep informing the controller of different simulators' hyper-parameters, including physical parameters of the constitutive model, resolution, and discretization choices. Moreover, since we treat the simulation resolution as a hyper-parameter, we are able to train the policy network using low-resolution simulators and transfer them to high-resolution ones with a small overhead, which significantly improves the sample efficacy. To generalize the controller over multiple tasks, we adopt hindsight experience replay (HER) [Andrychowicz et al. 2017], which reuses unsuccessful trajectory samples as surrogate optimal samples for a different task. Unlike the original HER where the task involves a single object, we allow a fluid spout to control multiple objects. To this end, we design a compact task representation. Our approach can achieve transferable control on complex multi-body multi-task problems without re-training. We highlight these new features on a row of 4 benchmarks illustrated in Figure 1.

2 RELATED WORK

We review prior works on the control of fluids, solid objects, as well as the design of general controllers.

2.1 Fluid Control

A large body of fluid control algorithms [McNamara et al. 2004; Pan and Manocha 2017a; Tang et al. 2021; Treuille et al. 2003] assume fully actuated dynamic systems where ghost forces can be injected anywhere and some algorithms [Pan and Manocha 2017a; Tang et al. 2021; Treuille et al. 2003] further assume single-phase flows. As a result, a short control horizon suffices and the dynamic system is differentiable. However, computing the gradient information is still costly. In parallel, researchers have also proposed non-physical fluid animation editing algorithms. Nielsen and Bridson [2011] proposed to simulate a large body of fluids at a low-resolution and switch to a high-resolution for fine details. [Thuerey 2016] proposes to synthesize new animations from existing data via physics-inspired interpolation. More recent works [Kim et al. 2019; Prantl et al. 2019; Xie et al. 2018] propose deep neural architectures to synthesize small-scale details from a dataset of examples. These methods are fast, but meanwhile, may loss some physical authenticity, especially at the boundary of interpolation. Recently, Li et al. [2019] and [Li et al. 2020] showed that end-to-end learning architectures can be trained to mimic fluid simulators, which is inherently amenable to control problems by providing analytical gradients. However, their results are limited to small-scale examples. Our method is complementary to all these techniques in that we do not modify the underlying physics model, while achieve control by only changing the boundary conditions. Similar to our method, recent researches [Hu et al. 2019c; Ma et al. 2018; Schenck and Fox 2018] are increasingly relying on data-driven algorithms to control the true dynamics of high-dimensional deformable bodies.

2.2 Controller Design for Solid Objects

Thanks to the low-dimension nature of near-rigid solid objects, their control and simulation algorithms typically run at real-time, which find many applications in the graphics community. Early works [Popović et al. 2003] control the pose of solid objects without considering contacts and collisions. More recent methods such as [Toussaint et al. 2020] jointly infer discontinuous contact events as well as continuous solid object poses. Solid control algorithms are most widely used in character animations. This direction of research evolved from using (partially) manually designed controllers [Si et al. 2015; Tan et al. 2011; Wang et al. 2012; Yin et al. 2007], contact implicit trajectory optimization [Coros et al. 2012; Pan and Manocha 2018a; Pan et al. 2019; Posa et al. 2014; Tan et al. 2012; Tassa et al. 2012], all the way to deep reinforcement learning [Bergamin et al. 2019; Min et al. 2019; Park et al. 2019;

Peng et al. 2018, 2021; Won et al. 2020; Yu et al. 2018], with a trend of minimizing the human intervention in controller design and parameterization. Learning-based techniques have also been used to synthesize animations without considering physical models, e.g., in [Starke et al. 2021]. Among these works, a closely related problem to ours is swimming creature control [Min et al. 2019; Pan and Manocha 2018b; Si et al. 2015] where characters utilize fluid contact forces to propel themselves. Instead, we assume unactuated near-rigid objects that are controlled indirectly using high-dimensional fluid bodies, which adds an additional layer of challenge.

2.3 Transfer Learning and Multi-Task RL

Transfer learning algorithms enable a controller to quickly adapt to new environments from small amounts of experiences. To this end, the gradient-based meta-RL methods [Finn et al. 2017; Mendonca et al. 2019] and the context-based meta-RL methods [Duan et al. 2016; Rakelly et al. 2019] assume that a controller can utilize a few steps of gradient decent or sample a small amounts of experiences online to generate task-specific contexts. These works find important applications in hardware robot control, when transferring from simulated to real environments. In our application, however, the context is defined by different simulation parameters (resolution, viscosity, density, etc.) and algorithms (smoothed particle hydrodynamics [Becker and Teschner 2007], particle-in-cell methods [Zhu and Bridson 2005], material point methods [Hu et al. 2018], etc.). Complementary to transfer learning, multi-task RL algorithms aim at a controller that performs reasonably well over a set of control objectives. To this end, task-based RL algorithms [Andrychowicz et al. 2017; Schaul et al. 2015; Zhao et al. 2019] learn a universal policy to solve multiple tasks. Schaul et al. [2015] generalises the state and task to learn a task-based policy. Andrychowicz et al. [2017] uses task-based experience replay method to tackle tasks with sparse reward signals. Rather than generalizing states and tasks, we propose a compact task representation to tackle multiple tasks.

3 PROBLEM FORMULATION

We use Figure 2 to illustrate the problem and the major components of our framework. In this showcase, tasks may involve maintaining balls in air and switching horizontal ball positions. The controller is informed of the encoded tasks, e.g., the latter task can be expressed by the ball index sequences (indices ordered in the horizontal direction) to be controlled and sent into the controller. From the index sequence, along with other fluid and object states, we train a policy network to generate control signals. The users are able to modify the simulation environments (e.g. doubling the simulator's resolution of discretization). The controller must recognize and adapt to these simulator changes on the fly. In this section, we first introduce the mathematical model of our coupled fluid-solid dynamics system, and then give an overview of our method for the transferable multi-task control problem.

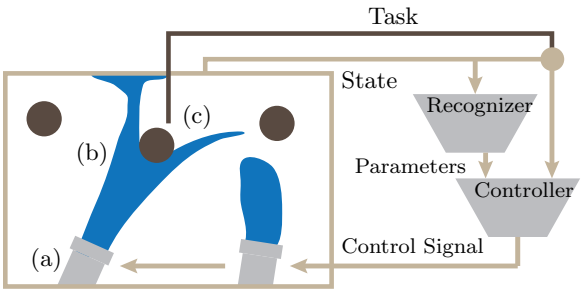


Fig. 2. In this showcase, the two fluid spouts (a) emit liquid (b) and push the three balls (c) to keep a dynamic force equilibrium. The controller is informed of the ball indices, i.e., the encoded task, as well as the current state of the coupled dynamic system. Given the state, we train a recognizer network to decode the current simulation parameters (resolution, simulator type, etc.), which is also forwarded to the controller.

Name (State)	Dimension in 2D/3D	Range
Position (q, c)	2/3	$(-\infty, \infty)$
Velocity (q, c)	2/3	$(-\infty, \infty)$
Orientation (q, c)	1/3	$(-\pi, \pi)$
Angular Velocity (q, c)	1/3	$(-\pi, \pi)$
Out-flow Flux (q only)	1/1	$(0, 1)$

Table 1. Components of q, c per rigid body. For controlled fluid spouts, we model their emission status as continuous out-flow fluxes within $(0, 1)$, which is interpreted as the cross-section area of emitted fluid columns.

3.1 Coupled Fluid-Solid Dynamics System

We consider two different fluid simulation algorithms, smoothed particle hydrodynamics (SPH) [Becker and Teschner 2007] and material point method (MPM) [Hu et al. 2018], and refer readers to these two papers for low-level technical details. On the high-level, we represent the state of our coupled system as a 3-tuple: $s \triangleq \begin{pmatrix} u & q & c \end{pmatrix}$, where u is the fluid's velocity field, q is the state of controlled solid objects (e.g. position, orientation, velocity, and outflux of fluid spouts), c is the concatenated states of all the passive, uncontrolled solid objects (e.g. balls in the water pool). In a 2D (resp. 3D) simulation domain, the dimension of c is $6n_s$ (resp. $12n_s$), the dimension of q is $7n_c$ (resp. $13n_c$), and the dimension of u (denoted as N) is simulator dependent (see Table 1 for detailed state encoding). For an SPH simulator, N is the number of fluid particles, while for a MPM simulator, N is the number of grid cells. Here n_s is the number of solid objects, which in our experiments is at most $n_s \leq 6$, n_c is the number of controlled solid objects, which in our experiments is at most $n_c \leq 2$. We represent the state of a rigid object as $c \triangleq \begin{pmatrix} x & w & \dot{x} & \dot{w} \end{pmatrix}$, where x is the position and w is the orientation. In our MPM experiments, N is orders of magnitude larger than that of the solid state c. A discretized physics simulator can then be denoted as a discrete transfer function: $s_{t+1} = f(s_t, a_t, z)$, where we use subscript t to indicate a variable at the tth time

instance and a_t is the action vector encoding all the control signals. Inspired by the usage of environmental variables in the context of meta-RL, we introduce a variable z to denote simulator parameters that remain constant throughout each simulation procedure, which is a major difference from [Ma et al. 2018]. Our considered environmental variables include grid resolution, solid density, fluid density, gravity, and the type of simulator.

3.2 Simulator-Transferable, Multi-Task Control

We use the standard problem definition of a Markov decision process (MDP) in continuous action spaces and follow definitions in [Haarnoja et al. 2018]. In a single-task setting, our control problem is defined by the tuple $\begin{pmatrix} S & A & p & r & \gamma \end{pmatrix}$, where $s \in S$ is the state space, $a \in A$ is the action space, $p(s_{t+1}|s_t, a_t)$ is the state-transition probability, which is derived by injecting noises into our discrete fluid simulator. The objective of RL is to maximize the following cumulative, bounded reward signal $r : S \times A \to \mathbb{R}$:

$$J = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) \right]. \tag{1}$$

Our policy is represented as a function $\pi : S \to A$ that maps each state to the corresponding optimal action, maximizing the expected return. Here $p(\tau) = p(s_0) \prod_{t=0}^{\infty} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$ is the distribution of trajectories, $(s_0, a_0, s_1, ...)$, under policy π with initial state distribution $p(s_0)$, and γ is the discount factor.

In a multi-task setting, we introduce a task space G where each $g \in G$ represents a sub-task (e.g., one of the ball index sequences in Figure 2). We further account for changing simulation environments by introducing a latent simulator feature space $z \in Z$. Inspired by [Rakelly et al. 2019], the controller can generalize to multiple environments by modeling z as a stochastic distribution during the training stage. As a result, our reward signal is augmented as a function $r : S \times A \times G \times Z \to \mathbb{R}$ and policy as a function $\pi : S \times G \times Z \to A$. Finally, our augmented objective function becomes:

$$J = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t, g, z) \right], \tag{2}$$

where $p(\tau) = p(s_0) \prod_{t=0}^{\infty} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t, g, z)$. We further improve the robustness of our controller via novel task representation and HER techniques (explained in Section 4.4 in detail). We always assume the number of available controllers is less than the number of objects to be controlled, i.e. $n_c < n_s$, which is a challenging setting where the controller must learn to choose objects to be controlled and switch between controlling multiple objects.

4 METHOD

In this section, we introduce the designs of our RL-based multi-body control method for both simulator-transferable and multi-task control. We first describe the network architecture in Section 4.1 and the efficient training baseline for the policy network in Section 4.2. We then use a context-based meta-RL technique [Rakelly et al. 2019] to encode simulator parameters, which involve parameters of the constitutive model and solver types. An efficient transfer technique is described in Section 4.3, where a sample-selection strategy during the training stage significantly improves the convergence rate. Thanks to the transfer technique, we can train the controller using a low resolution and transfer the control skills to a higher resolution. As illustrated in Table 2, our training resolution is $3-5$ times lower than the test-time resolution, saving over one order of magnitude of computational time. We further achieve multi-task learning by training a universal controller that can solve all tasks in a discrete set (Section 4.4). Key to the efficacy of our multi-task controller are the HER mechanism and a compact task representation that greatly reduces the training complexity.

4.1 Network Architecture

Our controller parameterization is illustrated in Figure 3. Fluid bodies are peculiar in that they have an extremely high-dimensional state u. Fortunately, the velocity field is highly smooth and correlated as mentioned in [Tompson et al. 2017], which means we can use dimension reduction techniques to obtain an effective low-dimensional representation. Indeed, the difference of resolution mainly affects the details of the fluid surface and small vortices, which do not serve an important role in liquid-solid force interactions compared to the bulk motion, which can be nicely captured in a down-sampled velocity field. Therefore, we pre-train a velocity encoder mapping u to a low-dimensional feature space $\psi(u)$. To train the velocity encoder, we first run our simulators using randomly sampled actions and record the velocity fields. We minimize the reconstruction loss to update the parameters of the velocity encoder, to which the same procedure was used in [Ma et al. 2018]. Furthermore, we find that, compared to prior velocity encoders where the last convolution layer is flattened and the encoded feature is computed through an additional fully connected layer, a Fully Convolutional Network (FCN) can better encode fluid features and bring better performance in downstream tasks. However, our domain size is anisotropic. In order to make the number of learnable parameters in FCN approximately the same, we use an anisotropic convolution kernel with a proportional size to the domain size. In summary, we denote the low-dimensional state as: $\bar{s} \triangleq \begin{pmatrix} \psi(u) & q & c \end{pmatrix}$, i.e., our observation space consists of a fluid velocity field, position and velocity information of both controlled and uncontrolled objects. We list the network parameters in Table 3 and Table 4, respectively.

Our main control module is an actor-critic framework, where an actor $\pi_\varphi(a|\bar{s}, z)$ maps the extended state $(\bar{s}, z)$ to a low-dimensional action a. The critic consists of one double-Q-network [Hasselt et al. 2016] and one value-network, which approximates the state-action- and state-value function, respectively. These two networks are used to guide the actor. Finally, after a full trajectory is simulated, we collect transfer

Example	Training-Time Resolution	Test-Time Resolution	Avg. N (Training)	Avg. N (Testing)	Training Time
Solid Scooping from Water Tank(2D)	$128 \times 128 \sim 192 \times 192$	512×512	49K	460K	30h
Targeted Solid Scooping(2D)	$128 \times 128 \sim 192 \times 192$	512×512	49K	466K	38h
Multi-Solid Balancing(3D)	$80 \times 48 \times 32 \sim 120 \times 72 \times 48$	$400 \times 240 \times 160$	15K	739K	54h
Multi-Solid Acrobatics(3D)	$80 \times 48 \times 32 \sim 120 \times 72 \times 48$	$320 \times 192 \times 128$	15K	932K	70h
Multi-Solid Music Player(3D)	$80 \times 48 \times 32 \sim 120 \times 72 \times 48$	$320 \times 192 \times 128$	9.3K	433K	74h

Table 2. Specifications of the training and testing environments. The benchmarks are measured using an MLS-MPM simulator and on a single Nvidia RTX2080ti GPU. Training resolutions significantly affect training time and is randomly sampled in a certain range. As a result, #particles used in the MPM solver varies drastically and we plot the average #particles throughout each training/testing. Training times are measured as wall-time from start to convergence.

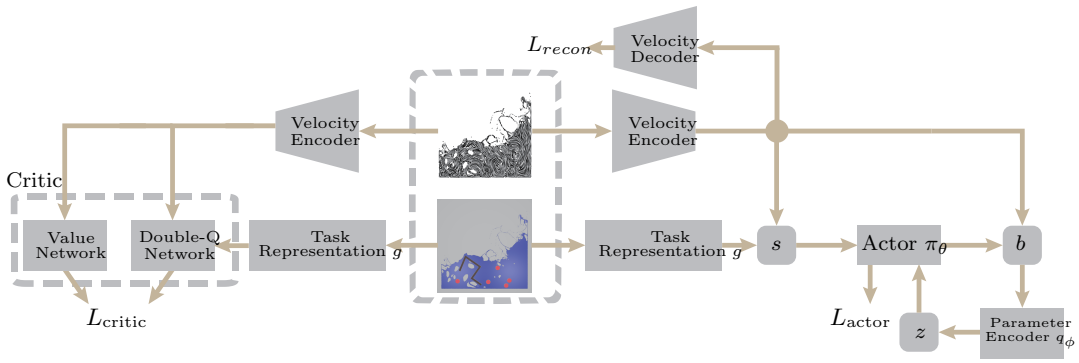


Fig. 3. We illustrate all the networks used by our architecture, including the fluid-velocity autoencoder, the parameter encoder, the double-Q-network, and the value-network. All the networks except for the velocity auto-encoder are MLPs, which take the velocity feature as inputs. We show the critic-architecture (resp. actor-architecture) on the left (resp. right) side of the example state.

tuples $b =< \bar{s}_t, a_t, \bar{s}_{t+1} >$, from which the simulator parameters are inferred using the parameter encoder. The modules mentioned above are all parameterized using MLPs without any weight-sharing. We illustrate the network architecture in Figure 3 and list the network parameters in Table 5.

4.2 Off-Policy Reinforcement Learning

Prior method [Ma et al. 2018] trained RL-based fluid controllers using an on-policy algorithm [Schulman et al. 2015], which iteratively updates controller parameter θ via sample-approximate of ∇J_θ. Such algorithms require new trajectory samples even for a small update of policy parameters, which largely limits their sample efficacy. On the other hand, an off-policy algorithm [Haarnoja et al. 2018] splits the responsibility between two networks: actor and critic. The critic network $Q_\theta(s_t, a_t, g, z)$ learns to approximate cumulative reward from a given state-action tuple. It can then be used as a surrogate objective for updating the actor without requiring new trajectory samples. As a result, off-policy algorithms are oftentimes more sample efficient, which is crucial in our applications where fluid simulation samples are highly costly to compute. Further, it has been shown [Hasselt et al. 2016; Silver et al. 2014] that training the actor and critic alternatively while fixing the target Q-network can significantly stabilize RL training and avoid over-estimation of value functions. Inspired by these findings, we choose to update our actor, critic, and parameter encoders in an alternating manner.

Layer	Kernel	Stride	#Filters	Activation
$conv_1$	(7,7)	2	64	LeakyReLU
$conv_2$	(7,7)	2	64	LeakyReLU
$conv_3$	(5,5)	2	128	LeakyReLU
$conv_4$	(5,5)	2	128	LeakyReLU
$conv_5$	(5,5)	2	128	LeakyReLU
$conv_6$	(2,2)	2	128	LeakyReLU
$conv_7$	(2,2)	2	64	LeakyReLU

Table 3. Parameters for 2D velocity encoder, which is symmetric to the 2D decoder.

Layer	Kernel	Stride	#Filters	Activation
$conv_1$	(3,3,3)	2	64	LeakyReLU
$conv_2$	(3,3,3)	2	64	LeakyReLU
$conv_3$	(3,3,3)	2	128	LeakyReLU
$conv_4$	(5,3,2)	1	128	LeakyReLU
$conv_5$	(5,3,2)	1	128	LeakyReLU
$conv_6$	(2,2,2)	1	64	LeakyReLU

Table 4. Parameters for 3D fluid encoder, which is symmetric to the 3D decoder.

Network	#Hidden Layers	#Neuron Per Layer	Activation
Actor	3	300	ReLU
Critic	3	300	ReLU
Parameter Encoder	3	200	ReLU

Table 5. Parameters for MLPs. Both the value network (represented as a MLP) and the double-Q network (represented as 2 MLPs) are part of our critic. Altogether, we use 3 separate MLPs of the same architecture to represent the critic.

Our policy network system is trained with the off-policy reinforcement learning algorithm: Soft Actor-Critic (SAC) [Haarnoja et al. 2018]. The critic network implements policy evaluation, which approximates the value function for the current state. The actor network, on the other hand, implements policy improvement, which obtains a better policy guided by the gradient of the critic. Finally, an experience replay buffer is used to collect history trajectories sampled with different policies. For every iteration, data is sampled from the replay buffer to alternately train the critic and the actor. Further, the SAC algorithm combines the optimization objective with a maximum entropy term to encourage exploration. This term further improves the versatility of the trained policy and is thus more favorable for our subsequent transfer process.

We extend the basic SAC method with the latent variable z, and our optimization objective has been described in Equation 2. Our critic network $Q_\theta(\bar{s}, a, g, z)$ encodes the value of action a under the state $\bar{s}$, the task vector g, and the simulator parameter z. Correspondingly, our actor is the policy $\pi_\varphi(a|s, g, z)$. Here θ and φ are parameters of the neural network. Since SAC requires stochastic policy, we redefine the action a be the sufficient statistics of a Gaussian distribution centered at $\pi_\varphi(a|s, g, z)$ with optimizable diagonal variance. The critic network can be updated by the TD-λ technique [Sutton and Barto 2018] with the following critic-loss:

$$L_{\text{critic}} = \mathbb{E}_{\substack{(s_t,a_t,s_{t+1})\sim D \\ z\sim q_\phi(z|s_t,a_t,s_{t+1})}} \left[Q_\theta(s_t, a_t, g, z) - (r + \gamma V(s_{t+1}, g, z))\right]^2, \tag{3}$$

where V is target network used by the double-Q-learning framework [Wang et al. 2016] and D is the replay buffer. The actor-loss can be written as a KL divergence between the policy and Q-value:

$$L_{\text{actor}} = \mathbb{E}_{\substack{(s_t,a_t,s_{t+1})\sim D \\ z\sim q_\phi(z|s_t,a_t,s_{t+1})}} \left[D_{KL}\left(\pi_\varphi(\cdot|s_t, g, z)\middle\| \frac{\exp(Q_\theta(s_t, \cdot, g, z))}{Z_\theta(s_t, g, z)}\right)\right], \tag{4}$$

where the function $Z_\theta(s_t, g, z)$ is used to normalize the distribution. We refer readers to [Haarnoja et al. 2018] for more details. In the above loss functions, we assume that the simulation environment parameter z is available from a parameter encoder q_ϕ introduced in the next section.

4.3 Controller Transfer

We inform our controller of simulator changes via a 20-dimensional latent variable z. These latent variables do not have physical meaning and are not trained in a supervised manner. Instead, we train a sub-network to infer z from sampled trajectories. After sampling a complete trajectory (using the same simulator parameter) in the training stage, we assume that z can be inferred from the transition tuples $b_t \triangleq (s_t, a_t, s_{t+1})$, which obeys a Gaussian distribution $q_\phi(z|b)$, where q_ϕ is the sufficient statistics of the Gaussian distribution inferred by the parameter encoder. However, the transition tuples, b, are subject to severe noise, which is caused by stochastic events of the fluid, including complex topological changes, vortices, etc. Eventually, such noise hinders the convergence of the sub-network training to a meaningful representation of the simulator parameters. To alleviate this issue, we choose to use only the first T tuples of each trajectory with a total length K as training inputs for this sub-network. In these early-stage tuples, fluids and solids start to move from their rest state where few stochastic events can happen. As a result, these motions potentially show the inherent characteristics of the simulator. In our experiments, we find this small set of tuples is enough for the sub-network to learn a good simulator parameter representation. We further limit the size of the sub-network to infer z from a single tuple b and represent the final distribution of z by combining the estimation of all T tuples as:

$$q_\phi(z|b_{1:T}) = \prod_{n=1}^{T} q_\phi(z|b_n). \tag{5}$$

We train ϕ and θ, φ in an interleaved manner. After updating the actor and critic network, we update ϕ using the following loss:

$$L(\phi; b) = L_{\text{KL}} + L_{\text{critic}}, \tag{6}$$

where the following KL divergence implements the information bottleneck and avoids overfitting:

$$L_{\text{KL}} = \beta D_{\text{KL}}(q_\phi(z|b))||p(z)). \tag{7}$$

Here $p(z)$ is the unit normal distribution. During the inference stage, we use Equation 6 to continually update the latent variable z. Specifically, for each newly-sampled trajectory, we collect the first T tuples and apply Equation 6 to derive $q_\phi(z|b_{1:T})$. The multiplicative nature of this formula allows more and more trajectories to be combined to get an increasingly accurate z estimation. Ultimately, the z-conditioned actor can be prepared to face a variety of new simulator parameters, without the need of re-training.

4.4 Multi-Tasking

In this section, we refer to a "task" as a single target or a set of similar sub-targets one want a solid object to achieve, for example, "moving a ball to a position x" can be considered as one task with all x in a small vicinity considered sub-tasks. We introduce a task-representation scheme that is able to handle multi-task control problems of this kind.

4.4.1 State Vector Mapping With Task Representation. When different solid objects are targeted at (assigned to) different tasks, Schaul et al. [2015] used a task vector g to encode the object-to-target assignment, which is combined with s to form a new state vector $\begin{pmatrix} s & g \end{pmatrix}$ as input to the actor network. However, this approach can lead to a combinatorial

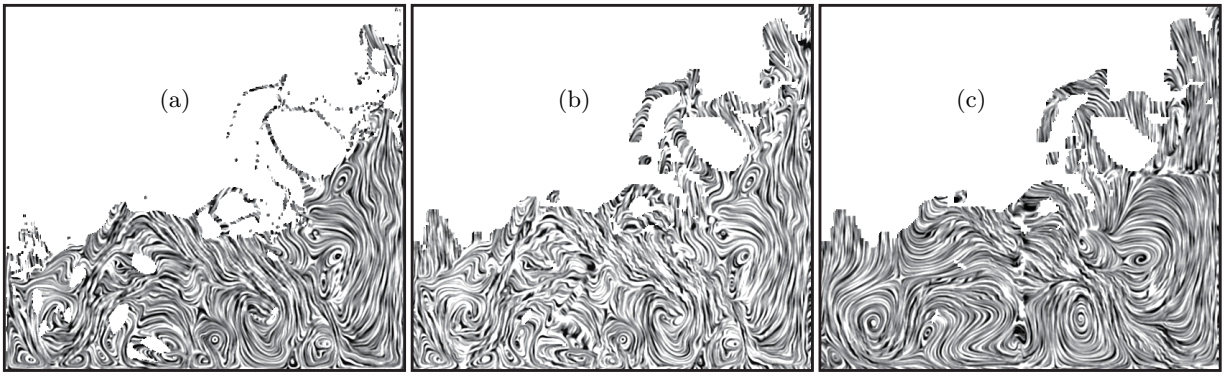


Fig. 4. Reconstruction of fluid velocity field by the autoencoder. (a) A velocity field extracted form a high-resolution simulation; (b) The same field sampled at a 0.25× resolution; (c) the velocity field reconstructed by the auto-encoder. By comparing (b,c), we see that the little vortices are lost but the bulk motion is largely preserved.

explosion in the task set. For example, scooping up 2 good balls out of other 8 bad ones can produce 45 possible task vectors. It is hard to train a universal controller for all these tasks via previous methods even with a considerable amount of sampling and training efforts. In our approach as illustrated in Figure 5, we assume that for each sub-task of a task, which might belong to a given set of goals for the solid objects, the composition of the goal set is fixed (e.g. there are exactly 2 out of 10 balls to be moved, but which two can vary). Therefore, we can pre-label each kind of sub-tasks, and rearrange the order of solid objects' state vectors, c, to reflect their sub-task labels (e.g. the 2 balls to be moved always appear first in the state vector). The order of good/bad balls in the state vector are not important, since switching objects of the same shape/type does not affect the simulator's state. This treatment thus significantly reduces the number of possible sub-tasks in the task space G as well as the sample complexity during the training stage (we still call $g \in G$ as a task vector that corresponds to a sub-task in the task space for compactness). The network only needs to learn a small set of sub-tasks or even a single task by making the representation invariant to switching. At the beginning of each trajectory, the user specifies the good/bad assignments g and we switch ball ordering of the state vector to always put the good balls in g first, eliminating g from the state vector. In practice, this switching scheme is implemented as the first layer of our policy network, which is intended to be a state transformation function that is neither learnable nor differentiable.

4.4.2 Hindsight Experience Replay. Control problems involving multiple objects and tasks oftentimes have sparse reward signals, where trajectories of positive rewards are rare and hardly sampled. We adopt the Hindsight Experience Replay (HER) to increase the number of positive samples. This method is based on the fact that we can treat failed cases (negative samples) of one given task as positive samples of another task by matching the goal of that task. In our approach, when we observe a negative sample, where we want one object

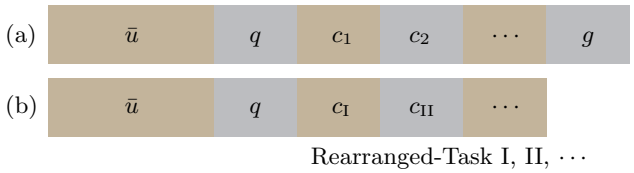


Fig. 5. The difference between task representation method [Schaul et al. 2015] (a) and our method (b). Schaul et al. [2015] does not change the order of solid objects' states but use a task vector g to reflect the task information. Instead, we rearranges objects' states by the task to reduce the state space's dimension.

to reach some goal position but end up having another object reaching that same position, we denote such sample as a potential sample. We then re-assign the task of the potential sample, re-map the state vector using our task representation, and insert the tuple into our replay buffer. Another case of a potential sample is when one object achieves a wrong sub-task. Although not all negative samples can be made positive, this technique significantly improves the rate of positive samples in the replay buffer, and increases the chances for sparse reward signals to be felt. In our experiments, we found that the performance of HER can be improved by considering an entire trajectory, instead of a single timestep. Specifically, if the cumulative reward of an entire trajectory increases after re-assigning the task, we insert all the tuples of a trajectory into the replay buffer. Otherwise, we discard the re-assigned trajectory.

4.5 Overall Algorithm

Algorithm 1 outlines the training of a transferable controller across a variety of simulators and tasks. For every iteration, we first infer the environment-relevant variable z to represent the current simulator, then we generate a trajectory with z-conditioned actor π_φ, and insert collected data into the replay buffer and update b. Then we try to change the current task to

ALGORITHM 1: Transferrable, Multi-Task Policy Training

A set of random simulators $\{\tau_i\}$, task set G, learning rates $\alpha_{1,2,3}$
Initial parameters θ, ϕ, φ, replay buffer $D \leftarrow \emptyset$
for *each iteration* **do**
  for *each* τ_i **do**
    Initialize $b^i \leftarrow \emptyset$
    Sample a sub-task g in G
    ▷Sample Trajectory
    for *t=1,...,K* **do**
      Sample $z \sim q_\phi(z|b)$
      Sample $a_t \sim \pi_\varphi(s_t, g, z)$
      Advance simulator and add
      $D \leftarrow D \cup \{(s_t, a_t, s_{t+1}, g, z, r_t)\}$
      $b^i \leftarrow b^i \cup \{(s_t, a_t, s_{t+1})\}$
    ▷Implement HER
    Sample a set of additional sub-tasks g' in G
    for *each* g' **do**
      if $\sum_{t=1}^{K} r_t(s_t, a_t, g, z) < \sum_{t=1}^{K} r'_t(s_t, a_t, g', z)$ **then**
        for *t=1,...,K* **do**
          $D \leftarrow D \cup \{(s_t, a_t, s_{t+1}, g', z, r'_t)\}$
  ▷Implement SAC
  for *sampled batch from* D **do**
    $\phi = \phi - \alpha_1 \bigtriangledown_\phi (L_{critic} + L_{KL})$
    $\theta = \theta - \alpha_2 \bigtriangledown_\theta L_{critic}$
    $\varphi = \varphi - \alpha_3 \bigtriangledown_\varphi L_{actor}$

find new trajectories whose total reward is higher for a multi-task problem. If any better trajectories are found, we insert them into the replay buffer too (essentially implementing HER). All networks are updated after data collection. For each sampled data batch from the replay buffer, we use the critic-loss (Equation 3) to update the Q-value network, the actor-loss (Equation 4) to update the policy network, and the combined KL- and critic-loss (Equation 6) to update the parameter encoder.

5 EXPERIMENTS & EVALUATION

We evaluate the performance of our method on a set of challenging 2D and 3D benchmarks. All the examples are trained in low-resolution simulators and tested in high-resolution ones (Table 2). A typical 2D-controller (resp. 3D-controller) training takes about 35 (resp. 70) hours with a Nvidia RTX2080Ti GPU. It takes millions of time steps to train a 3D controller in our benchmarks. Without using our approach, it would take several months to train the controller directly in the resolution of $400 \times 240 \times 160$, which is impractical. We implement the simulators with TaiChi programming language [Hu et al. 2019b]. We use two types of physical solvers, MPM and SPH. The MPM (resp. SPH) solver is implemented by MLS-MPM method [Hu et al. 2018] (resp. WCSPH [Becker and Teschner 2007]). Our training consists of two passes, i.e. first a pre-training pass of the autoencoder for fluid-velocity field encoding, then the main training pass. During the pre-training of the autoencoder, we use $L_{\text{recon}} \triangleq \mathbb{E}_u \left[\|u - E(D(u))\|^2\right]$ and a dataset of 80000 velocity field samples to optimize the velocity encoder for both 2D and 3D control tasks with uniform random actions. Here we denote $E(\cdot)$ as the encoder function and $D(\cdot)$ as the decoder function. It is known that the autoencoder can be made over-fitting robust if it outputs a feature distribution instead of a feature vector, which is known as variational autoencoder (VAE). However, we found that VAE requires more learnable parameters to represent both the mean and variance of a distribution, which is not suitable for our small dataset size, so we choose to use plain autoencoder. A variety of simulator parameters are considered, including resolution, gravitational coefficient, type of solver, Young's modulus in the MPM solver, viscosity and vorticity in the SPH solver. Figure 4 shows a reconstructed velocity field by the learnt autoencoder network, which largely preserves the down-sampled bulk motion.

We consider two kinds of controllers, spoons and water spouts, both of which can be considered boundary conditions to the fluid simulator, while the large bulks of fluid bodies move according to the true dynamics. In 2D environments, we define $q = \begin{pmatrix} x & w \end{pmatrix}$ for each controller. All the controls are realized by applying force and torques to q, and we denote $\dot{q} = \begin{pmatrix} \dot{x} & \dot{w} \end{pmatrix}$ and $\ddot{q} = \begin{pmatrix} \ddot{x} & \ddot{w} \end{pmatrix}$ as its velocity and acceleration. For water spouts in 3D environments, we add constraints to limit their motions within the horizontal X-Z plane and rotations to the vertical X-Y plane. As a result, q still has three components, i.e. $q = \begin{pmatrix} x_X & x_Z & w_{XY} \end{pmatrix}$, which takes the same form as that in 2D. Finally, we add a regulation to the control action, suppressing excessive control signals by introducing thresholds to $q, \dot{q}, \ddot{q}$ so that $q \in B_q$, $\dot{q} \in B_{\dot{q}}$ and $\ddot{q} \in B_{\ddot{q}}$. We will use subscript (resp. superscript) in the equations to index controlled (resp. uncontrolled) objects. We will summarize our training parameters and scene setting parameters in Appendix A. In all the following examples, at the inference stage, for each scene setting, our learned meta-RL network first samples 2 trajectories to update the z variable as described in Section 4.3.

5.1 Solid Scooping from Water Tank

Our first, 2D benchmark is inspired by prior robotic research [Pan and Manocha 2017b; Schenck and Fox 2017] on liquid transfer. We formulate a more challenging task that requires extreme precision, where five small balls are placed in water and a spoon is placed above the water surface at first, and the goal is for the controlled spoon to scoop up multiple solid balls as fast as possible, as illustrated in Figure 6. This is achieved by limiting the trajectory length and designing a reward function that continually produces positive values when a ball falls inside the spoon. There can be several variants of such precision manipulation tasks. To start with, we have one spoon initially outside the water and five balls placed statically in water and our goal is to scoop up as many balls as possible. We use the following reward function to model this task:

$$r(s,a) = \omega_x f(x_{\text{spoon}} - x^*_{\text{spoon}}) + \mathcal{I}\left(\|x_{\text{spoon}} - x^*_{\text{spoon}}\| > d_0\right)\left[\omega_{\dot{x}} f(\dot{x}_{\text{spoon}}) + \omega_{\dot{w}} f(\dot{w}_{\text{spoon}})\right]$$

Training Method	Solver(Stage)	Resolution	#Particles	Gravity	Young's Module	Density Ratio	Spoon Shape	Avg. #Balls Scooped
Meta-RL	MLS-MPM	512×512	460.8K	$10 \sim 70$	$500 \sim 1000$	$0.8 \sim 1.6$	Parallelogram	1.72
Standard RL With Random Simulators	MLS-MPM	512×512	460.8K	$10 \sim 70$	$500 \sim 1000$	$0.8 \sim 1.6$	Parallelogram	1.13
Standard RL With Fixed Simulator	MLS-MPM	512×512	460.8K	$10 \sim 70$	$500 \sim 1000$	$0.8 \sim 1.6$	Parallelogram	1.17
Manually Design	MLS-MPM	512×512	460.8K	$10 \sim 70$	$500 \sim 1000$	$0.8 \sim 1.6$	Parallelogram	0.65
Meta-RL	WCSPH	\	54K	$30 \sim 50$	\	$0.5 \sim 1.0$	Parallelogram	1.47
Standard RL With Random Simulators	WCSPH	\	54K	$30 \sim 50$	\	$0.5 \sim 1.0$	Parallelogram	0.85
Standard RL With Fixed Simulators	WCSPH	\	54K	$30 \sim 50$	\	$0.5 \sim 1.0$	Parallelogram	0.73
Manually Design	WCSPH	\	54K	$30 \sim 50$	\	$0.5 \sim 1.0$	Parallelogram	0.68

Table 6. We compare the online performance of a controller trained using meta-RL, standard RL and manually designed policy. To this end, two controllers are trained using meta-RL and standard RL with random MPM simulators, where we randomly sample parameters to generate different simulators in the training stage, including the resolution, gravity, fluid-to-solid density, Young's module, and we change the shapes of the spoon to be smoothly deforming between a series of parallelograms. The other two controllers are trained using standard RL with a fix simulator or manually designed. During the inference stage, we randomly choose 20 different simulators whose parameter scopes are shown in this table, and profile the average number of balls scooped within 3 trials using each unseen simulator. Our meta-RL-based approach significantly outperforms manually designed policies or those trained via the standard RL in transferring tasks.

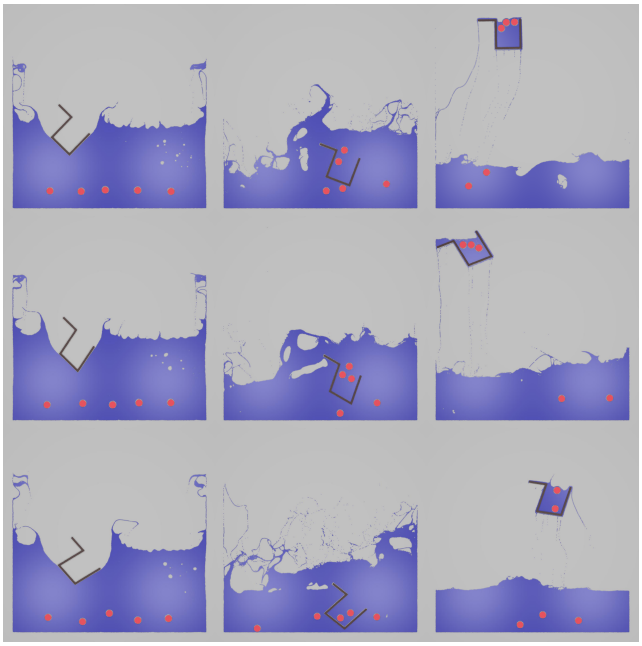
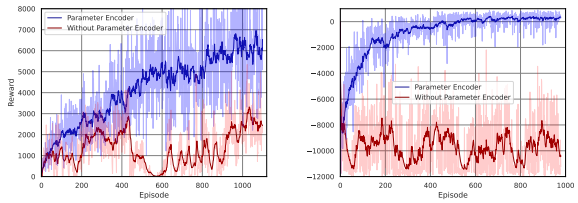


Fig. 6. Illustration of the solid scooping benchmark, where the goal is for the spoon to pick up as many balls as possible. We demonstrate three trials of the same trained network on different spoon shapes in each row.

$$f(\bullet) \triangleq n^2 \exp(-\| \bullet \|^2), \tag{8}$$

where n is the number of balls that end up inside the spoon, $\mathcal{I}(\bullet)$ is the indicator function, and x^*_{spoon} is the target position. The terms $f(\dot{x}_{\text{spoon}})$ and $f(\dot{w}_{\text{spoon}})$ will encourage the spoon to hold a stationary pose and keep the scooped balls from spilling when the spoon center is within a distance d_0 from the target position.

We use this benchmark to demonstrate and analyze the ability of simulator transfer. During the training phase, we generate sample trajectories from a series of low-resolution MPM simulations with randomly-set physical parameters,



(a) Solid Scooping (b) Solid Balancing

Fig. 7. RL optimizer suffers from a slow progress (a) or stalling (b) without meta-learning and parameter encoder when using different simulators in training, which is because the policy gradients from different simulators may cancel each other. Using our training method, the optimizer quickly distinguishes gradients by their simulator parameters and distill useful information from these gradients to improve cumulative rewards.

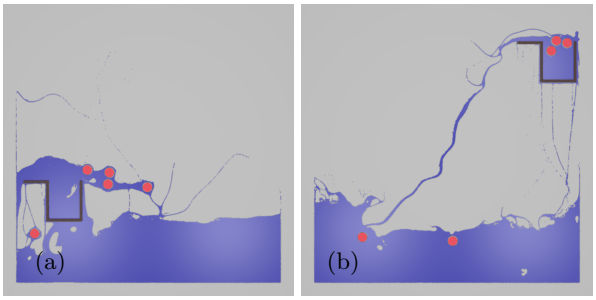


Fig. 8. We illustrate a typical failure case of manually designed baseline controller (a) as compared with our policy (b) in the targeted scooping task.

including gravity, fluid density, Young's modulus, resolutions ($128 \times 128 \sim 192 \times 192$), and spoon shapes. Figure 7(a) demonstrates the reward change during training with/without our parameter encoder. Without the help of the parameter encoder (i.e. using standard SAC), the optimizer exhibits a low training efficiency due to the conflicting policy gradients introduced by various simulator environments. On the

contrary, our meta-learning scheme allows the policy to distinguish the gradients by their simulators and quickly improve the cumulative rewards. We also perform another ablation study comparing our method with standard SAC, where the standard SAC method is used to train the policy with a fixed set of physical parameters. The policy is then applied to other simulation environments. In Table 6, we run standard SAC with both random and fixed physical parameters during training, and compare their results with our meta-RL-based method. All policy networks are trained under a low resolution, sampled uniformly between 128×128 and 192×192 (the fixed setting is only trained under 128×128), and tested at a high resolution of 512×512. Other parameters vary within the same range in both training and testing stages. The meta-RL based method largely outperforms the fine-tuned standard RL ones. As expected, using random simulators with standard RL cannot help generalizing the policy, echoing the analysis in Figure 7.

To highlight the complexity of the multi-solid scooping task, we further conduct an ablation study and compare our policy with a manually designed baseline controller. Many prior works in robotics such as [Pan and Manocha 2017b] manually design parametric scooping actions for liquid transfer tasks, which involve moving the scoop towards the object and then moving back up. As illustrated in Figure 8, we evaluate both policies in the scooping problems whose goals are to get as many balls as possible. We randomly generate 20 scooping cases with different sets of environmental parameters and evaluate three trials for each case. For the final scooped number of ball, our policy achieves an average number of 1.72, while the baseline controller only achieves 0.65. Detailed parameter settings can be found in Table 6. One possible reason behind this phenomena is that, in contrast to single-object tasks, the strict incompressibility makes the motions of different balls strongly coupled in a complex way, and the fluid information is essential to success.

We also test the above trained controllers on a WCSPH simulator (with $\sim$ 50K particles), which is another unseen simulator during the training stage. At test time, a random fluid-solid density ratio is chosen, and we report the statistical results, including physical parameters setting and average scooped number of balls in Table 6. The results show that our approach successfully transfers the learned policy to new simulation environments.

5.2 Targeted Solid Scooping

Our second variant of the ball-scooping task requires a higher level of manipulating precision. Instead of scooping up balls without discrimination, our goal in this case is to scoop the "good ones" (in red) but leave behind the "bad ones" (in black). In addition to being precise, we also require the controller to handle changing good ball assignments online, leading to a multi-task setting. Settings for this benchmark are the same as our first 2D benchmark, but in the beginning, the user can arbitrarily specify a certain number (2 for our benchmark) of balls as the "good ones". To model this task, we use the same reward signal as Equation 8 except for the function $f(\bullet)$ replaced by the following function:

$$f(\bullet) = \omega_{\text{right}} \left[\sum_{i=1}^{n_s} \mathcal{I}\left(x^i \text{ in spoon} = x^i \text{is good}\right)\right]^2 \exp(-\| \bullet \|^2) - \omega_{\text{wrong}} \left[\sum_{i=1}^{n_s} \mathcal{I}\left(x^i \text{ in spoon} \neq x^i \text{is good}\right)\right]^2 \exp(-\| \bullet \|^2). \tag{9}$$

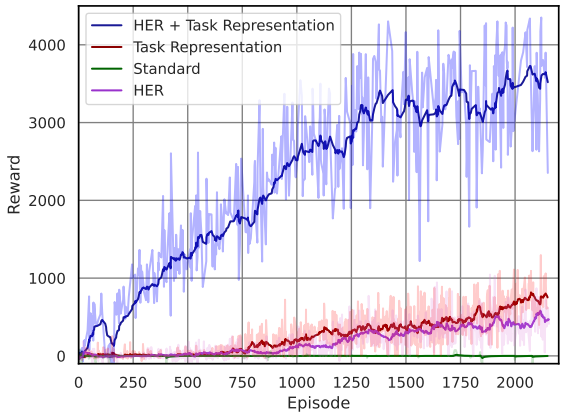


Fig. 9. A comparison highlighting the effect of HER and our task representation techniques in the targeted solid scooping task using 8 black balls. This is a challenging multi-task setting with sparse reward signals. Our results show that both HER and task representation are indispensable for successful training.

In this benchmark, as illustrate in Figure 12, we apply both the HER and task representation techniques as described in Section 4.4.1. We evaluate the performance of our multi-task learning technique in this benchmark with two target red balls and a number of bad black balls. We note that even with a small number of balls (e.g., using 4 bad balls in Figure 12(a)), there is already a 15-sized set of object-task combinations, and the complexity rapidly grows when the number of bad ball increases (e.g., in Figure 12(b), the 8 bad ball case has a 45-sized set of combinations). In such tasks the reward signal is extremely sparse. We compare the results with/without HER and our task representation technique in Figure 9. As expected, our HER and task representation technique significantly improves the optimized cumulative reward. In Table 7, we profiled the rate of successfully and accurately picking up target balls with different number of bad balls. The task difficulty also increases rapidly when the number of possible object-task combination reaches a certain level.

Number of Bad Balls	4	6	8	10
Success Rate	83	65	60	10

Table 7. Success rate of the targeted ball scooping task when there are $4, 6, 8$, or 10 bad balls and 2 good balls. We consider a trial successful when the good balls are all scooped without any bad balls in the spoon. We find that the task difficulty increases rapidly when the number of possible object-task combinations reaches a certain level.

5.3 3D Multi-Solid Balancing

Our second, 3D benchmark is inspired by the 2D ball balancing benchmark [Ma et al. 2018], but our version is much more complex in that we use two water spouts to control three solid balls to keep balanced in a 3D domain. Since the number of controlled objects is more than the number of controllers, our controller has to learn to coordinate between different targets both temporally and spatially. We consider a ball successfully balanced when it is confined in a cuboid region without hitting the boundary of that region (as illustrated in Figure 10), which is modelled by the following reward signal:

$$r(s,a) = \sum_{i=1}^{n_s} \left[\omega_x \exp(-\left\| x^i_{\text{ball}} - x^{i*}_{\text{ball}} \right\|^2) + \omega_{\dot{x}} \exp(-\left\| \dot{x}^i_{\text{ball}} \right\|^2) \right] - \omega_{hit} \sum_{i=1}^{n_s} \mathcal{I}\left(x^i_{\text{ball}} \text{ hits boundary} \right), \quad (10)$$

where the x^i_{ball}-related term penalizes position mismatch and $\dot{x}^i_{\text{ball}}$-related term encourages static balances. During the training stage, once an object touches the boundary of the cuboid region, it will be bounced back. We add an indicator of how many balls hit the boundary at the current timestep, which is multiplied by a non-positive coefficient as an additional penalty. This bouncing boundary of the cuboid region is only used to limit the range of ball motions during the training stage, so as to reduce the training complexity. We remove the boundary during the test time and our controller can empirically always confine the balls within given cuboid regions.

A 3D control task with such a huge state space as fluid simulators requires a gazillion of samples. However, with our off-policy RL and meta-RL enabling low-resolution training, we are able to finish the policy optimization within 60 hours. Figure 7(b) illustrates the training curve with or without the parameter encoder.

We find the solid balancing task sensitive to simulation environmental parameters. Because the controller must keep the solid in a dynamic force equilibrium, any under- or over-estimation of forces can result in failure. We utilize this sensitivity to highlight the necessity of meta-RL. Note Figure 7 has already shown that the standard RL using random physical parameters in training is not an effective way of policy generalization. So in Figure 11, we compare our policy against a standard SAC baseline policy, where both policies are trained at a low resolution, with the meta-RL network using random physical parameters and the standard RL using fixed physical parameters. The total trajectory numbers used in the training stages are the same for the two approaches. We then test how well the two trained policies transfer to a higher resolution by evaluating both policies in 20 trajectories with different resolutions. We find that our policy can keep the balls balanced for a longer time period then the SAC baseline at a higher resolution, as summarized in Table 8. The baseline approach performs better at its training resolution because it effectively observes far more trajectories at that resolution during training. However, the baseline quickly degrades facing unseen higher resolutions. In comparison, our meta-RL based policy stably applies to these unseen environments.

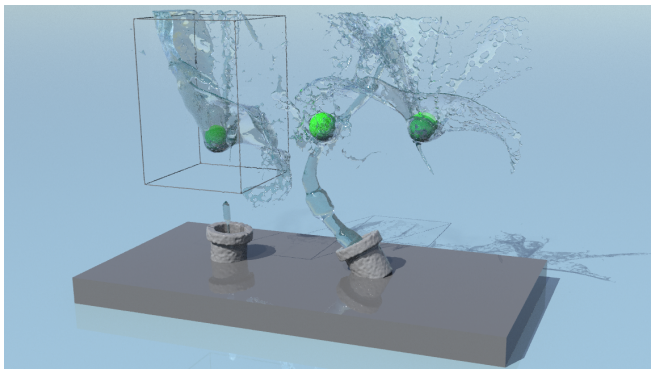


Fig. 10. Illustration of the 3D multi-solid balancing benchmark. The task is to confine the balls in a given cuboid region (illustrated by the wire frame). We successively balance the balls with two water spouts below using our trained network.

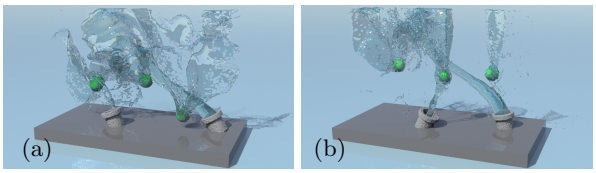


Fig. 11. We illustrate a typical failure case of policy without meta-RL (a) as compared with our policy (b) in the 3D multi-solid balancing task.

Training Method \ Resolution	$80 \times 48 \times 32$	$120 \times 72 \times 48$	$400 \times 240 \times 160$
Meta-RL	348	357	353
Standard SAC	456	360	216

Table 8. We profile the average number of frames in which the policy can keep the balls balanced. We compare two policies, one trained using meta-RL with random parameters (with random resolutions ranging from $80 \times 48 \times 32$ to $120 \times 72 \times 48$), and the other one is trained using standard RL with fixed parameters (with a fixed resolution at $80 \times 48 \times 32$). The performance of meta-RL approach stays approximately the same when scaled to unseen high-resolution environments.

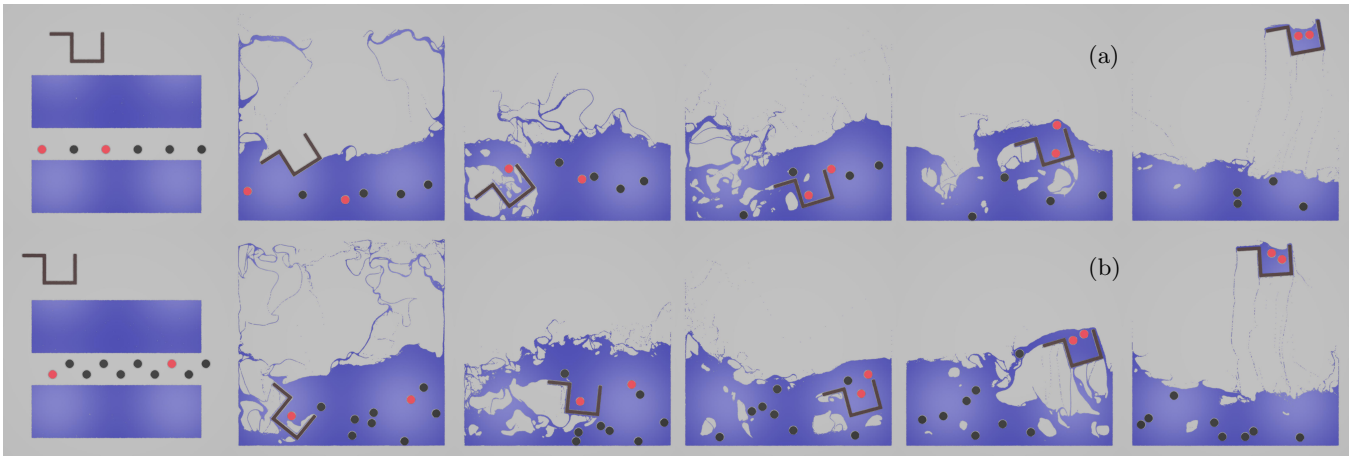


Fig. 12. A controller test of the targeted solid scooping task at a resolution of 512×512. Our controller scoops all the red balls and keeps out the black ones. (a): 2 target red balls among 4 black balls. (b): 2 target red balls among 8 black balls.

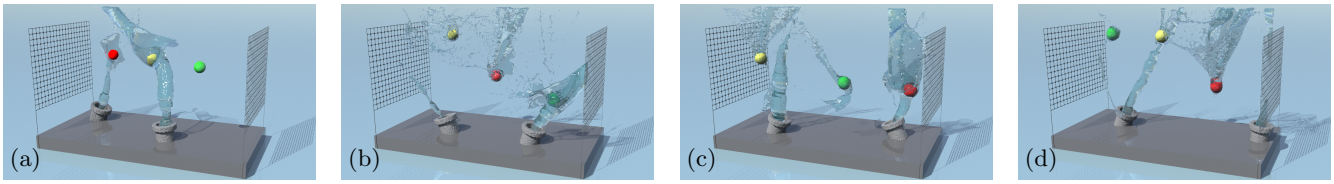


Fig. 13. Illustrations of multi-solid juggling benchmark. We initialize the three balls at different positions, the positions of two adjacent balls are then exchanged according to the user's online instructions: (a) initial positions; (b) switching positions of yellow and red balls; (c) switching positions of red and green balls; (d) switching positions of yellow and green balls.

5.4 3D Multi-Solid Juggling

Our third, 3D benchmarks is a multi-task, dynamic extension of the second benchmark, where we use two water spouts to control three solid balls to transpose horizontally in air, while keeping them from falling on the ground. During each manipulation, three small balls are initialized in air from left to right at their balancing target positions, and the user can arbitrarily assign two balls with target positions to be exchanged, while the third ball should stay near its original position as much as possible. This is also a multi-task control problem, as the transposed balls should be specified by the users online. We then divide the balls into three categories based on the relations between ball position and its target position: moving left, moving right and standing still, corresponding to expected horizontal velocities $[\dot{x}^{i*}_{\text{ball}}]_{\parallel} \in \{-0.5, 0.5, 0\}$, respectively. The target velocities are kept unchanged within a user-specified number of time-steps before re-deciding the ball categories. We use the following reward function to model this task:

$$r(s,a) = \omega_x \sum_{i=1}^{n_s} \exp(-\left\|[x^i_{\text{ball}}]_{\perp} - [x^{i*}_{\text{ball}}]_{\perp}\right\|^2) + \omega_{\dot{x}} \sum_{i=1}^{n_s} \exp(-\left\|[\dot{x}^i_{\text{ball}}]_{\parallel} - [\dot{x}^{i*}_{\text{ball}}]_{\parallel}\right\|^2) - \omega_{\text{hit}} \sum_{i=1}^{n_s} \mathcal{I}\left(x^i_{\text{ball}} \text{ hits scene boundary}\right), \tag{11}$$

where $[\bullet]_{\parallel,\perp}$ represents horizontal and vertical components of a vector. We demonstrate this benchmark in Figure 13.

5.5 Multi-Solid Music Player

Our last benchmark is the 3D extension to the music-player example of [Ma et al. 2018], where we set up a 3D music player that uses two water spouts to drive three balls in a cuboid scene. The balls are controlled to hit seven note keys (Do Re Mi Fa Sol La Ti, aligned along the x-axis) according to the input music script. For each manipulation, the user specifies an arbitrary note sequence, and the controlled water spout will push the balls to hit the notes in turn at a given frequency (e.g. quartertones). The complexity of this benchmark lies in that the controller must drive a proper ball and move it to meet the critical timing requirements according to the online-specified music script, where faraway note keys may be sequentially played.

We boil the task down to a combination of two simpler tasks. On observing the next key note in the script, we define one task to drive a proper target ball to hit the key note and another task to push the other two balls from the target ball along the x- and z-axes as far as possible. The purpose of this decomposition is twofold. First, we avoid collisions between balls and prevent balls from hitting a wrong key note. Second, by disseminating the three balls, it would

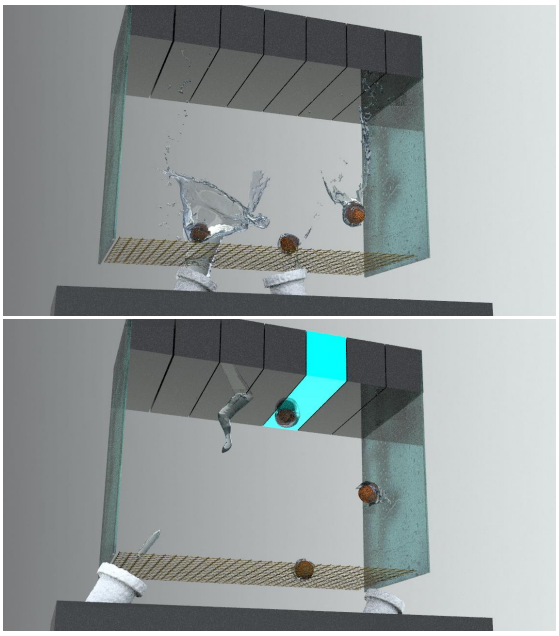


Fig. 14. Illustration of the 3D music player benchmark. Two spouts control three balls by jetting water in a 3D space, driving them to hit the key notes above according to the input music script. A strainer mesh is set to catch the dropped balls without affecting the flow of water. Our learned controller can play multiple music pieces completely unseen after the training stage.

be easier to find the proper ball with suitable distance and angle to hit the next key note, wherever it is.

We choose the next proper ball in the following way. First, we define a neighborhood around the next key note using a threshold of distance along the x-axis. If there are any balls within this neighborhood, we choose the one with lowest altitude as the target ball. Otherwise, we choose the ball with the minimal x-distance to the next key note. In practice, we further consider the balls' moving direction, and use the predicted position of a ball after a short time period (5 time steps) for the ball-selection. Our reward function thus consists of the following three parts:

$$
\begin{aligned}
r(s,a) =& r_{\text{over}} + r_{\text{hit}} + r_{\text{sep}} \qquad (12)\\
r_{\text{sep}} \triangleq& -\sum_{i=1}^{n_s} \mathcal{I}\left(i \text{ is not target}\right)\left[\omega_x \|x^i_{\text{ball}} - x^{i*}_{\text{ball}}\|\right]\\
r_{over} \triangleq& -\omega_{\text{over}} \sum_{i=1}^{n_s} \mathcal{I}\left(i \text{ is target}\right)\\
& \mathcal{I}\left(x^i_{\text{ball}} \text{ hits key note}\right)\mathcal{I}(|t-t^*| > \Delta t),
\end{aligned}
$$

where we set a hitting time range of Δt. Each note in the script has a target time denoted as t^* and each target ball must hit the correct key note within $[t^* - \Delta t, t^* + \Delta t]$. Failure to meet the timing requirement would receive a penalty of w_{over}. If the timing requirement is met, we use the following r_{hit} to reward a more accurate hit:

$$
\begin{aligned}
r_{\text{hit}} =& w_{\text{right}}\mathcal{I}\left(x^i_{\text{ball}} \text{ hits right key note}\right) -\\
& w_{\text{wrong}}\mathcal{I}\left(x^i_{\text{ball}} \text{ hits wrong key note}\right)\|x^i_{\text{ball}} - x^{i*}_{\text{ball}}\| -\\
& w_t|t-t^*|. \quad (i\text{th ball is the target ball})
\end{aligned}
$$

Here we grant a constant, positive reward for a correct hit, while we scale the penalty by distance from the target key note for a wrong hit. The last term encourages a more accurate timing.

In the training stage of [Ma et al. 2018], they first randomly sample a sequence of notes and then keep them constant for the controller to learn a proper control policy for the sequence. This will need a lot of different random sample sequences for a universal policy network to be trained. When the music script is specified online, the performance of [Ma et al. 2018] drops significantly in our 3D music player setting due to their bad generalization ability. On the contrary, during the training stage, once a key note is hit, we set randomly the next target key note for the controller. Adopting our task representation and the HER techniques, our network achieves good performance after 74h of training. Once such a universal controller is trained, we can use it to play literally any music script. In our experiments, the average correct hit rate is above 70%, and it can reach 100% in some simple-music playing trials. Most incorrect hits involve miss-hit onto adjacent notes or early/late hits near the time threshold. We provide two pieces of melodies played by our controller in the supplemental video.

6 CONCLUSIONS & LIMITATIONS

We present a learning-based method to control both 2D and 3D fluid-directed multi-body systems. Our controller is represented as deep neural networks and trained using an efficient off-policy reinforcement learning algorithm. Our method features meta-RL framework allowing the controller to transfer to different simulators with a small amount of extra samples. Furthermore, high-resolution controls can be achieved with fast training on low-resolution simulators. Finally, a compact task representation is designed for solving multiple control tasks without the combinatorial explosion of the task space.

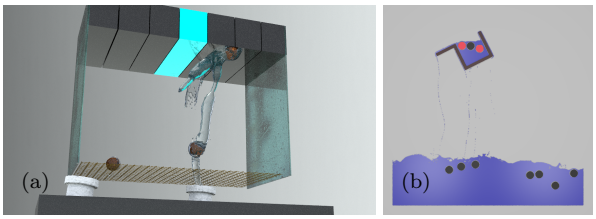


Fig. 15. We illustrate two mistakes made by our policy: hitting the wrong note key (a) and scooping the wrong ball (b).

One limitation of our meta-RL framework is the possibility of failure, especially when a new environment differs too much from training-stage environments. Due to the random nature of the RL algorithm and the complexity of fluid-solid interactions, a trained network does not guarantee success in each single run, even when the training converges to a high average reward, as shown in Figure 15(a) where the ball can hit a nearby wrong key during playing. The limited-length trajectory may also result in failure in certain cases as illustrated in Figure 15(b), where the spoon occasionally scoops a wrong ball early on but fails to shake it out within the period. Another limitation is that many hyper-parameters of RL training are still manually chosen, which significantly relies on trial and error. We have also found that

the learned policy is relatively sensitive to some parameters (e.g., number of trajectory samples, learning rate, and hidden layer size). The task complexity also affects the performance of the policy network. Taking the music-player for example, if we would like to further hold the balls in 3D space, preventing them from falling on the mesh, then failure cases significantly increase, where the balls cannot hit the right key at the right time. Currently these two tasks cannot be well-accomplished simultaneously using a small number of fluid spouts. Improving the success rate in such complicated tasks will be the focus of our future research.

A ARCHITECTURE PARAMETERS

In Table 9 and Table 10, we list all the parameters used in our benchmarks. During training, we sample from 20 simulators with different parameters for 2D benchmarks, and 10 simulators with different parameters for 3D benchmarks. We further list various parameters used by the training algorithm, which include the timestep size, number of timesteps per trajectory, number of trajectories per iteration, number of training steps per iteration.

Parameters	Value
Reward Coefficients ($\omega_x, \omega_{\dot{x}}, \omega_w$)	2,0.25,0.25
Target Position Distance (d_0)	0.2
Target Solid Scooping Coefficient ($\omega_{\text{right}}, \omega_{\text{wrong}}$)	1,0.4
Position Threshold ($\text{Bq}_{min}, \text{Bq}_{max}$)	[0,0,-3],[1,1,3]
Velocity Threshold ($\text{B}\dot{q}_{min}, \text{B}\dot{q}_{max}$)	[-4,-4,-12],[4,4,12]
Acceleration Threshold ($\text{B}\ddot{q}_{min}, \text{B}\ddot{q}_{max}$)	[-50,-50,-150],[50,50,150]
Timestep Size	0.012(s)
Number of Training Steps per Iteration	2000
Number of Timesteps per Trajectory	150
Number of Trajectories per Iteration	15

Table 9. Parameters for 2D benchmarks.

Thresholds for Controllers	Value
Position Threshold ($\text{Bq}_{min}, \text{Bq}_{max}$)	[0.2,0.1,1.0],[1.4,0.54,2.2]
Velocity Threshold ($\text{B}\dot{q}_{min}, \text{B}\dot{q}_{max}$)	[-20,-10,-20],[20,10,20]
Acceleration Threshold ($\text{B}\ddot{q}_{min}, \text{B}\ddot{q}_{max}$)	[-1500,-1000,-1500],[1500,1000,1500]
Multi-Solid Balancing	Value
Reward Coefficients ($\omega_x, \omega_{\dot{x}}, \omega_{\text{hit}}$)	2.5,1.5,25
Timestep Size	0.004(s)
Number of Timesteps per Trajectory	1000
Number of Trajectories per Iteration	1
Number of Training Steps per Iteration	1000
Multi-Solid Juggling	Value
reward coefficients ($\omega_x, \omega_{\dot{x}}, \omega_{\text{hit}}$)	1.5,3,25
Timestep Size	0.004(s)
velocity-keeping time-steps	200
Number of Timesteps per Trajectory	200
Number of Trajectories per Iteration	8
Number of Training Steps per Iteration	1000
Music Player Parameters	Value
Reward Coefficients ($\omega_x, \omega_{\text{over}}, \omega_{\text{right}}, \omega_t, \omega_{wrong}$)	0.05,10,35,0.6,15
Timestep Size	0.006(s)
Number of Timesteps per Trajectory	1000
Number of Trajectories per Iteration	2
Number of Training Steps per Iteration	1500

Table 10. Parameters for 3D benchmarks.

ACKNOWLEDGMENTS

This work is supported by the Open Project Program of State Key Laboratory of Virtual Reality Technology and Systems, Beihang University (No.VRLAB2021A04).

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc., Long Beach, California, USA. https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf

Markus Becker and Matthias Teschner. 2007. Weakly Compressible SPH for Free Surface Flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '07)*. Eurographics Association, Goslar, DEU, 209217.

Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: Data-Driven Responsive Control of Physics-Based Characters. *ACM Trans. Graph.* 38, 6, Article 206 (nov 2019), 11 pages. https://doi.org/10.1145/3355089.3356536

Stelian Coros, Sebastian Martin, Bernhard Thomaszewski, Christian Schumacher, Robert Sumner, and Markus Gross. 2012. Deformable Objects Alive! *ACM Trans. Graph.* 31, 4, Article 69 (jul 2012), 9 pages. https://doi.org/10.1145/2185520.2185565

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. 2016. RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning. arXiv:1611.02779 [cs.AI]

Raanan Fattal and Dani Lischinski. 2004. Target-Driven Smoke Animation. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 441448. https://doi.org/10.1145/1015706.1015743

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, Sydney, Australia, 1126–1135. https://proceedings.mlr.press/v70/finn17a.html

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. 2018. Meta-Reinforcement Learning of Structured Exploration Strategies. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 53075316.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholm, Sweden, 1861–1870. https://proceedings.mlr.press/v80/haarnoja18b.html

Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona) *(AAAI'16)*. AAAI Press, Phoenix, Arizona USA, 20942100.

Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. 2018. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019a. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 201.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019b. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–16.

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. 2019c. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*. IEEE, IEEE, Montreal, Canada, 6265–6271.

Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. 2019. Transport-Based Neural Style Transfer for Smoke Simulations. *ACM Trans. Graph.* 38, 6, Article 188 (nov 2019), 11 pages. https://doi.org/10.1145/3355089.3356560

Yunzhu Li, Toru Lin, Kexin Yi, Daniel Bear, Daniel Yamins, Jiajun Wu, Joshua Tenenbaum, and Antonio Torralba. 2020. Visual Grounding of Learned Physical Models. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daum III and Aarti Singh (Eds.). PMLR, Vienna, Austria, 5927–5936. https://proceedings.mlr.press/v119/li20j.html

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. 2019. Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids. In *ICLR*. Open Publishing, New Orleans, LA, USA.

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

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid control using the adjoint method. *ACM Transactions On Graphics (TOG)* 23, 3 (2004), 449–456.

Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. 2019. Guided Meta-Policy Search. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., Vancouver, Canada. https://proceedings.neurips.cc/paper/2019/file/d324a0cc02881779dcda44a675fdcaaa-Paper.pdf

Sehee Min, Jungdam Won, Seunghwan Lee, Jungnam Park, and Jehee Lee. 2019. SoftCon: Simulation and Control of Soft-Bodied Animals with Biomimetic Actuators. *ACM Trans. Graph.* 38, 6, Article 208 (nov 2019), 12 pages. https://doi.org/10.1145/3355089.3356497

Michael B. Nielsen and Robert Bridson. 2011. Guide Shapes for High Resolution Naturalistic Liquid Simulation. *ACM Trans. Graph.* 30, 4, Article 83 (jul 2011), 8 pages. https://doi.org/10.1145/2010324.1964978

Zherong Pan and Dinesh Manocha. 2017a. Efficient solver for spacetime control of smoke. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1.

Zherong Pan and Dinesh Manocha. 2017b. Feedback motion planning for liquid pouring using supervised learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Vancouver, BC, Canada, 1252–1259. https://doi.org/10.1109/IROS.2017.8202300

Zherong Pan and Dinesh Manocha. 2018a. Active Animations of Reduced Deformable Models with Environment Interactions. *ACM Trans. Graph.* 37, 3, Article 36 (aug 2018), 17 pages. https://doi.org/10.1145/3197565

Zherong Pan and Dinesh Manocha. 2018b. Active animations of reduced deformable models with environment interactions. *ACM Transactions on Graphics (TOG)* 37, 3 (2018), 1–17.

Zherong Pan, Bo Ren, and Dinesh Manocha. 2019. GPU-Based Contact-Aware Trajectory Optimization Using a Smooth Force Model. In *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '19)*. Association for Computing Machinery, New York, NY, USA, Article 4, 12 pages. https://doi.org/10.1145/3309486.3340246

Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. 2019. Learning Predict-and-Simulate Policies from Unorganized Human Motion Data. *ACM Trans. Graph.* 38, 6, Article 205 (nov 2019), 11 pages. https://doi.org/10.1145/3355089.3356501

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. 2017. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.

Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. 2021. AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control. *ACM Trans. Graph.* 40, 4, Article 144 (jul 2021), 20 pages. https://doi.org/10.1145/3450626.3459670

Jovan Popović, Steven M. Seitz, and Michael Erdmann. 2003. Motion Sketching for Control of Rigid-Body Simulations. *ACM Trans. Graph.* 22, 4 (Oct. 2003), 10341054. https://doi.org/10.1145/944020.944025

Michael Posa, Cecilia Cantu, and Russ Tedrake. 2014. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research* 33, 1 (2014), 69–81. https://doi.org/10.1177/0278364913506757 arXiv:https://doi.org/10.1177/0278364913506757

Lukas Prantl, Boris Bonev, and Nils Thuerey. 2019. Generating Liquid Simulations with Deformation-aware Neural Networks. arXiv:1704.07854 [cs.GR]

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. 2019. Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 5331–5340. https://proceedings.mlr.press/v97/rakelly19a.html

Karthik Raveendran, Nils Thuerey, Chris Wojtan, and Greg Turk. 2012. Controlling Liquids Using Meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Lausanne, Switzerland) *(SCA '12)*. Eurographics Association, Goslar, DEU, 255264.

Karthik Raveendran, Chris Wojtan, Nils Thuerey, and Greg Turk. 2014. Blending Liquids. *ACM Trans. Graph.* 33, 4, Article 137 (July 2014), 10 pages. https://doi.org/10.1145/2601097.2601126

Syuhei Sato, Yoshinori Dobashi, Theodore Kim, and Tomoyuki Nishita. 2018. Example-based turbulence style transfer. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–9.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. 2015. Universal Value Function Approximators. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 1312–1320. https://proceedings.mlr.press/v37/schaul15.html

Connor Schenck and Dieter Fox. 2017. Visual closed-loop control for pouring liquids. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, Singapore, 2629–2636. https://doi.org/10.1109/ICRA.2017.7989307

Connor Schenck and Dieter Fox. 2018. SPNets: Differentiable Fluid Dynamics for Deep Neural Networks. In *Proceedings of The 2nd Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 87)*, Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto (Eds.). PMLR, Stockholm, Sweden, 317–335. https://proceedings.mlr.press/v87/schenck18a.html

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 1889–1897. https://proceedings.mlr.press/v37/schulman15.html

Weiguang Si, Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. 2015. Realistic Biomechanical Simulation and Control of Human Swimming. *ACM Trans. Graph.* 34, 1, Article 10 (dec 2015), 15 pages. https://doi.org/10.1145/2626346

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 32)*, Eric P. Xing and Tony Jebara (Eds.). PMLR, Bejing, China, 387–395. https://proceedings.mlr.press/v32/silver14.html

Andrew Spielberg, Allan Zhao, Yuanming Hu, Tao Du, Wojciech Matusik, and Daniela Rus. 2019. Learning-in-the-loop optimization: End-to-end control and co-design of soft robots through learned deep latent representations. *Advances in Neural Information Processing Systems* 32 (2019), 8284–8294.

Sebastian Starke, Yiwei Zhao, Fabio Zinno, and Taku Komura. 2021. Neural Animation Layering for Synthesizing Martial Arts Movements. *ACM Trans. Graph.* 40, 4, Article 92 (jul 2021), 16 pages. https://doi.org/10.1145/3450626.3459881

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. 2011. Articulated Swimming Creatures. In *ACM SIGGRAPH 2011 Papers* (Vancouver, British Columbia, Canada) *(SIGGRAPH '11)*. Association for Computing Machinery, New York, NY, USA, Article 58, 12 pages. https://doi.org/10.1145/1964921.1964953

Jie Tan, Greg Turk, and C. Karen Liu. 2012. Soft Body Locomotion. *ACM Trans. Graph.* 31, 4, Article 26 (jul 2012), 11 pages. https://doi.org/10.1145/2185520.2185522

Jingwei Tang, Vinicius C. Azevedo, Guillaume Cordonnier, and Barbara Solenthaler. 2021. Honey, I Shrunk the Domain: Frequency-aware Force Field Reduction for Efficient Fluids Optimization. *Computer Graphics Forum* 40, 2 (2021), 339–353. https://doi.org/10.1111/cgf.142637 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.142637

Yuval Tassa, Tom Erez, and Emanuel Todorov. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Vilamoura, Algarve, Portugal, 4906–4913. https://doi.org/10.1109/IROS.2012.6386025

Nils Thuerey. 2016. Interpolations of smoke and liquid simulations. *ACM Transactions on Graphics (TOG)* 36, 1 (2016), 1–16.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2017. Accelerating Eulerian Fluid Simulation With Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, Sydney, Australia, 3424–3433. https://proceedings.mlr.press/v70/tompson17a.html

Marc Toussaint, Jung-Su Ha, and Danny Driess. 2020. Describing Physics For Physical Reasoning: Force-Based Sequential Manipulation Planning. *IEEE Robotics and Automation Letters* 5, 4 (2020), 6209–6216. https://doi.org/10.1109/LRA.2020.3010462

Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. 2003. Keyframe Control of Smoke Simulations. *ACM Trans. Graph.* 22, 3 (jul 2003), 716723. https://doi.org/10.1145/882262.882337

Jack M Wang, Samuel R Hamner, Scott L Delp, and Vladlen Koltun. 2012. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–11.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 1995–2003. https://proceedings.mlr.press/v48/wangf16.html

Jungdam Won, Deepak Gopinath, and Jessica Hodgins. 2020. A Scalable Approach to Control Diverse Behaviors for Physically Simulated Characters. *ACM Trans. Graph.* 39, 4, Article 33 (jul 2020), 12 pages. https://doi.org/10.1145/3386569.3392381

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. 2018. tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–15.

KangKang Yin, Kevin Loken, and Michiel Van de Panne. 2007. Simbicon: Simple biped locomotion control. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 105–es.

Wenhao Yu, Greg Turk, and C. Karen Liu. 2018. Learning Symmetric and Low-Energy Locomotion. *ACM Trans. Graph.* 37, 4, Article 144 (jul 2018), 12 pages. https://doi.org/10.1145/3197517.3201397

Yunbo Zhang, Wenhao Yu, C Karen Liu, Charlie Kemp, and Greg Turk. 2020. Learning to manipulate amorphous materials. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–11.

Rui Zhao, Xudong Sun, and Volker Tresp. 2019. Maximum Entropy-Regularized Multi-Goal Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 7553–7562. https://proceedings.mlr.press/v97/zhao19d.html

Yongning Zhu and Robert Bridson. 2005. Animating sand as a fluid. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 965–972.