Adaptable Pouring: Teaching Robots Not to Spill using Fast but Approximate Fluid Simulation

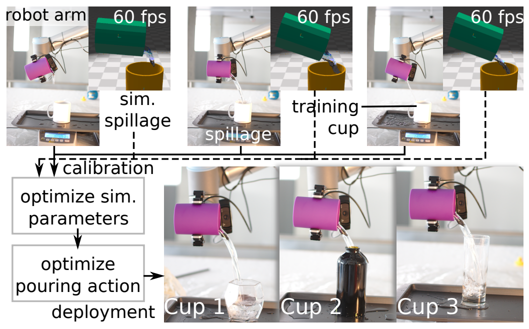
Tatiana López Guevara
Heriot-Watt University, University of Edinburgh
t.l.guevara@ed.ac.uk

Nicholas K. Taylor
Heriot-Watt University
n.k.taylor@hw.ac.uk

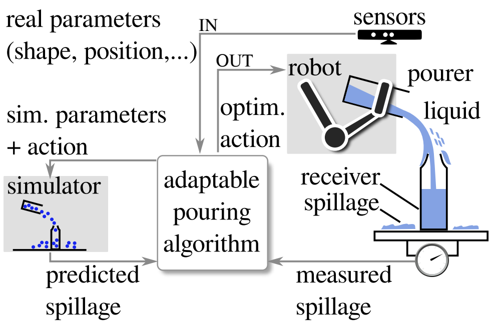
Michael U. Gutmann
University of Edinburgh
michael.gutmann@ed.ac.uk

Subramanian Ramamoorthy
University of Edinburgh
s.ramamoorthy@ed.ac.uk

Kartic Subr
University of Edinburgh
k.subr@ed.ac.uk



(a) pouring adapted to new cups



(b) system overview

Figure 1: We use fast, approximate fluid simulation in conjunction with real measurements to adaptively choose robotic actions that minimize spillage while pouring liquids into novel containers.

Abstract: Humans manipulate fluids intuitively using intuitive approximations of the underlying physical model. In this paper, we explore a general methodology that robots may use to develop and improve strategies for overcoming manipulation tasks associated with appropriately defined loss functions. We focus on the specific task of pouring a liquid from a container (pourer) to another container (receiver) while minimizing the mass of liquid that spills outside the receiver. We present a solution, based on guidance from approximate simulation, that is fast, flexible and adaptable to novel containers as long as their shapes can be sensed. Our key idea is to decouple the optimization of the parameter space of the simulator from the optimization over action space for determining robot control actions. We perform the former in a training (calibration) stage and the latter during run-time (deployment). For the purpose of this paper we use pouring in both stages, even though separate actions could be chosen. We compare four different strategies for calibration and three different strategies for deployment. Our results demonstrate that fast fluid simulations are effective, even if they are only approximate, in guiding automatic strategies for pouring liquids.

Keywords: Fluid simulation, Pouring

1 Introduction

The manipulation of liquids using autonomous robots opens doors to a plethora of exciting applications across diverse fields such as medicine, construction, the service industry, manufacturing pipelines, etc. In this paper, we address a specific manipulation task – teaching a robot to autonomously transfer all of the liquid in one container to another with minimum spillage. Our solution adapts to different containers without having to be re-trained.

Capturing the physics of fluids is both a complex modelling problem and one that usually results in sophisticated mathematical models. The resulting models capture a non-stationary process and solutions to the governing equations are often unstable. Despite being a deterministic process, the instability of solutions introduces unpredictability and hampers repeatability. It is therefore impractical to rely on precise simulations for making predictions about the flow of liquids. The use of high-fidelity fluid simulations would require a large computational budget as well as precise sensors. The inherent non-stationarity of the fluid simulation process, together with its uncountable state space, pose problems for generalized data-driven approaches. Low-fidelity simulations are feasible, due to their speed, but have other problems such as model mismatch since parameters to the simulator may not match physical quantities such as viscosity or density. We marshal evidence in this paper to demonstrate the utility of approximate fluid simulation in guiding pouring tasks.

Recently, there has been much interest in robotic manipulation of liquids [1, 2]. The general approach has been to couple simulation with motion control, using some form of sensing (thermal cameras [3], stereo cameras [4], etc.) to provide feedback. To make the problem tractable, other simplifications such as a representational quadratic curve for liquid pouring trajectories [5] have been made. Some methods take a data-driven approach, under the assumption that the robots have been trained with the specific containers used [1], or that the pouring containers have narrow spouts.

In this paper, we propose a two-stage adaptable pouring algorithm with the goal of minimizing spillage. Our approach adapts to novel containers at run time without requiring re-training. We assume that the geometry (shape) of the containers is known, determined by external sensors. In a first "calibration" stage, we map parameters to a fast simulator (NVIDIA FleX) to those of a specific liquid (water) using a pre-defined set of *actions* such as angular velocity of the pouring container and its maximum rotation angle. Then, during a "deployment" stage, we explore a potentially different action space to strategically transfer the liquid from one container to another while minimizing spillage. We demonstrate that fast simulation can be used to develop effective strategies for this pouring task.

2 Related work and contribution

The manipulation of fluids by autonomous robots requires an understanding of the physics of fluids and techniques in machine learning to develop innovative manipulation strategies.

Fluid simulation Physical models for fluids may be derived in many ways [6], depending on the accuracy, speed or stability required. One approach considers fluids as particles on a lattice that move along a discrete set of directions. This *Lattice-Boltzmann method* lends itself to easy parallelization and is well suited for fluid simulation at small scales but is inefficient to simulate coarse flows. Another considers the fluid as a continuum of particles and derives balance equations accounting for Newton's second law, stress and pressure locally within the fluid volume. These *Navier-Stokes* equations can can then be solved to estimate the velocity field by sampling at various points (Eulerian) or by gathering velocities of particles (Lagrangian) advected carefully. Contributions of particles may be interpolated (smoothed particle hydrodynamics [7]) based on the theory of integral interpolants using kernels that approximate a Delta function. Approximation using particles can also be used along with an Eulerian scheme (semi-Lagrangian) to produce a more stable result [8]. None of the above methods is straightforward to extend for two-way coupling between solids and liquids in a mixed simulation with rigid bodies as well as fluid particles.

Position-Based Dynamics (PBD): A notable exception to the aforementioned issue, is a fast but approximate method that handles two-way coupling by introducing dynamics based on the interactions of particles purely based on their positions [9]. PBD is used in video games for simulating liquids. We use this unified particle physics framework to model the behaviour of rigid bodies and fluids [10] in real-time. However, since this model is approximate and does not map directly to the physics of fluids, we need to accommodate for the model mismatch using a calibration stage.

Physical scene understanding There has been a lot of interest recently on explaining human judgements via *intuitive physical* simulations [11, 12, 13] that probabilistically account for uncertainty. This has also been explored for fluids [12] using a particle-based simulation model. Their experiments confirmed that despite being inaccurate, human judgment was correlated to their approximate model for probabilistic prediction. There have also been some models for qualitiative prediction [14] that agree with human judgement on which of two given liquids with different perceptual viscosities would pour out of a container first. The question of whether we use simulation models or emulation through our daily interaction with liquids remains open. Intuitive physics has also been used to infer latent properties of objects colliding with other objects [13] or being dropped in a liquid [15].

Model-free methods Another class of models learn directly from data. Either be passively, using data collected offline [16], or actively, where a robot builds an internal physics model from its own experience [17] or exploration [18]. Work on rigid bodies have also shown promising results by combining the expressive power of neural networks with the structure of a simulator [19, 20]. However, model-free methods are challenging for application in scenarios with fluids. In [2] neural networks were used to detect a liquid while being poured directly from visual input but generalization to different scenes and containers is still a problem. End to end learning was also used in [21] to learn a predictive model of flow but was only tested in 2D with very simple boundary conditions. Other methods exploit knowledge about physics while designing the learning solution. e.g. using physically based constraints [22], using regression forests to estimate the acceleration of fluid particles [23], or replacing the pressure projection step with a neural network [24]. However, they are either not real time or evidence of how well they perform on liquids with different properties was not shown.

Inference schemes The fidelity of the simulator depends on the choice of a number of free parameters. The statistics and machine learning literature provide principled methods (e.g. [25, 26]) for learning the parameter values of simulators from data. The general idea is to define a discrepancy measure between the simulated and observed data, and to identify the parameter values that yield small discrepancies. The search is typically computationally intensive because gradient information is not available. Bayesian optimisation [27] has recently been shown to effectively accelerate the search [28].

Robots and fluids Learning to pour can be done by demonstration using parametrization with respect to the perceived cup [29], or purely from haptic data [30], but they either ignored fluid dynamics or were not tested on real liquids. However, their experiments showed that during the pouring phase, humans tend to keep the tilting rate constant which is one of our assumptions in this work. Other approaches incorporate fluid-like behaviour via simulation to learn or guide poruing policies [31, 32, 5, 1]. However [31] was tested only on granular materials and in [32] the liquid is simulated as a set of rigid body spheres controlled by a bouncing parameter, which was manually tuned. Pan et al. [5] used a hybrid particle-mesh fluid simulator, but it took around one hour to compute a single pass of the pouring scene being modelled. This approach was later extended [1] to reduce the computational burden by training a parabolic heuristic approximation, but it still needs to be pre-trained offline for all combinations of container shapes and liquid materials. Recent work demonstrates the use of closed-loop fluid simulation along with a real robot [3] towards a pouring task. The approach used a thermal camera to track heated water being poured and then performed an image-space comparison (2D projection space) to predictions from an SPH [7] simulation engine. The method is fast, but cannot be generalized since it requires liquids to be heated thus changing their flow properties.

Summary Our approach is goal-based and does not require a state-space model for fluid simulation. Instead we directly parameterize the objective (minimizing spillage) over the space of actions that a robot can perform. We evaluate our approach using a real robot and a digital weighing scale to measure spillage. We use a PBD simulator to guide the actions performed by the robot. We do not require retraining on specific containers. We assume that the geometry of the containers and their relative position are known (or can be sensed).

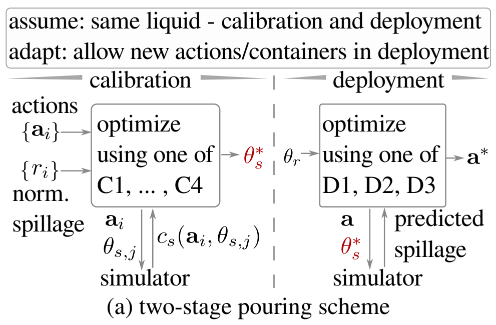
Contributions The key contributions of this paper are that we:

1. demonstrate that approximate simulation models (PBD) are useful for guiding pouring tasks;
2. overcome the model mismatch of PBD without performing cumbersome inference on physical quantities such as viscosity, density, etc.;
3. propose a goal-based discrepancy function between simulation and real measurements (normalized discrepancy between spilled liquid in our case);
4. develop a method that can optimize over a space of control actions that may not be in the training set, e.g. we train using angular action parameters and evaluate using spatio-angular control;

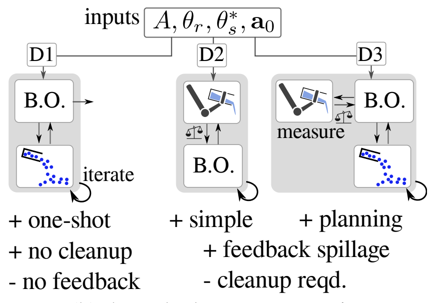
3 Using fast and approximate simulation for adaptable pouring

3.1 Definitions, notation and assumptions

We define *actions* as variables that specify control of the robot. e.g. angular velocity, spatial position, or maximum rotation angle. We define *simulation parameters* as input variables that need to be supplied to the simulator. e.g. for NVIDIA FleX, they could be number of particles, viscosity, cohesion, or buoyancy. We use $\mathbf{a} \in A$ to denote actions, $\mathbf{a}^{(i)}$ to denote the i^{th} dimension of the action and $\mathbf{a}^*$ to represent an optimal action. A represents the set of all actions. Similarly $\theta \in \Theta$, $\theta^{(i)}$ and θ^* for parameters. We use subscripts to qualify whether actions (and parameters) are real or whether they are within simulation, e.g. $\theta_r^{(i)}$ is the i^{th} dimension of the real world parameter θ_r, Θ_s is the set


assume: same liquid - calibration and deployment
adapt: allow new actions/containers in deployment
calibration
deployment
actions
$\{\mathbf{a}_i\}$
$\{r_i\}$
norm.
spillage
optimize using one of C1, ... , C4
θ_s^*
θ_r
optimize using one of D1, D2, D3
$\mathbf{a}^*$
$\mathbf{a}_i$
$\theta_{s,j}$
$c_s(\mathbf{a}_i, \theta_{s,j})$
simulator
$\mathbf{a}$
θ_s^*
predicted spillage
simulator

(a) two-stage pouring scheme



(b) three deployment strategies

Figure 2: (a) We first identify optimal simulation parameters (calibration) for simulated spillage to match real spillage in training. We use the calibrated parameters to optimize over action spaces to minimize spillage. (b) We propose three strategies D1-D3 for optimizing over the action space.

of all simulation parameters, and so on. If the fluid simulation model was perfect then $\Theta_r = \Theta_s$. For models, such as the one used in this paper, this is not the case since the parameters of the simulation span a lower dimensional space and may not correspond to physical quantities. We assume that real parameters θ_r, such as the shape of the containers, can be known. Figure 2 presents an overview of our method along with the notation used.

3.2 Problem formulation

We define a cost function $C_r : A_r \times \Theta_r \rightarrow \Re$ that measures the liquid spilled outside the receiver. The cost is defined as the mass of spilled fluid (in grams) after pouring. The overall goal of this paper is to determine an action $\mathbf{a}^*$ that minimizes the cost $C_r(\mathbf{a}, \theta_r)$. While we can assume that we do know θ_r, we do not know the functional form of $C_r(\mathbf{a}, \theta_r)$, which makes direct minimization of the cost impossible.

We use a two-step approach to solve this problem: In a training step, we calibrate a simulator so that we can use it to predict the spillage which we would expect for a certain action (similar to [3, 22]). Then, during the deployment stage, we will use the simulator to find the optimal action efficiently. Since real parameters are assumed to be known, and fixed during calibration, we will drop their usage to simplify notation.

3.3 Calibration

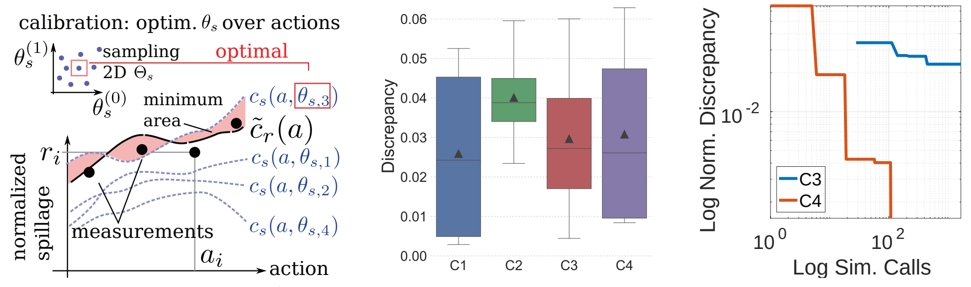
The simulated spillage $C_s : A_s \times \Theta_s \rightarrow \Re$ is the number of particles that fall outside the receiver after pouring. Since they do not measure the same quantity as the real spillage C_r (controllable parameters on the approximate simulator are constrained and do not correspond to real physical properties), we normalize the costs by the total mass (k_m) and the total number of particles (k_s) of the liquid to be poured out of the pouring container in reality and simulation, respectively, and work with the normalized quantities $c_r = C_r/k_r$ and $c_s = C_s/k_s$.

During the calibration phase, a set of n actions $\mathbf{a}_i$ are performed by the (real) robot and the resulting (normalized) spillage r_i is recorded for each action. Due to the robot's imprecision, we assume that the measured spillage equals $c_r(\mathbf{a}_i)$ subject to additive noise, $r_i = c_r(\mathbf{a}_i) + \eta_s$. Let the spillage produced by the simulator for the same actions $\mathbf{a}_i$ and some simulation parameters θ_s be s_i.

In order to calibrate the simulator, we introduce a discrepancy $\Delta(\mathbf{a}, \theta_s) = \mid c_r(\mathbf{a}) - c_s(\mathbf{a}, \theta_s) \mid$ that measures the absolute difference between the normalized cost functions for a common action $\mathbf{a} \in A_r \cap A_s$. Since we are only concerned with actions that can be simulated as well as executed by a robot, henceforth we will use A in place of $A_r \cap A_s$. The discrepancy needs to be computed over the space of actions, which can either be computed using only the observed actions or by considering all actions. We present two optimization problems that are based on the discrepancy $\Delta(\mathbf{a}, \theta_s)$

$$O_1 : \quad \underset{\theta_s \in \Theta_s}{\operatorname{argmin}} \sum_{i=1}^{n} |r_i - c_s(\mathbf{a}_i, \theta_s)| \qquad \text{and} \qquad O_2 : \quad \underset{\theta_s \in \Theta_s}{\operatorname{argmin}} \int_A |c_r(\mathbf{a}) - c_s(\mathbf{a}, \theta_s)| \, g(\mathbf{a}) \, \mathrm{d}\mathbf{a}$$

For O_2, $g(\mathbf{a})$ is a prior distribution over the space of actions. Furthermore, while we do not know $c_r(\mathbf{a})$, we can use the observed samples $\{(\mathbf{a}_i, r_i)\}_{i=1..n}$ to build an approximation $\tilde{c}_r \approx c_r$.



(a) 1D action space A (b) evaluation C1-C4 (c) decay rate of error

Figure 3: (a) Illustration of calibration, over a 2D parameter space and a 1D action space, to find θ_s^*. (b) Evaluation of the quality of θ_s^* from four calibration schemes, using real executions on test actions applied to a robot. The box represents lower to upper quartile values of the data with a line at the median, a triangle at the mean and whiskers show the range of the data. Ideally, we expect zero discrepancy. (c) For increasing number of simulations (to estimating θ_s^*), the error drops more rapidly for C4 (compared to C3).

We propose (and evaluate below) a total of four different calibration strategies: namely grid search for O_1 (C1 and C2), Bayesian Optimization (B.O.) for O_1 (C3), and B.O. for O_2 (C4).

C1, C2. Grid search We generated samples $\theta_{s,j} \in \Theta_s$ over the parameter space and computed the cost (summation term in O_1) for each of the samples. We used uniform sampling for C1 and stratified (jittered) sampling for C2. θ_s^* is the sample $\theta_{s,j}$ that results in the smallest cost. The computation of the cost function requires n simulations to be performed at each of the $\theta_{s,j}$.

C3. Bayesian optimisation using measured actions C1 and C2 evaluate all actions without taking the observed performances into account. It is reasonable to assume that actions which are similar to previously unsuccessful ones will not perform much better. Strategies that aim to choose actions based on previous performance might be much more efficient. Thus, we used standard Bayesian optimisation to minimize O_1, thereby identifying samples $\theta_{s,j} \in \Theta_s$ in successive iterations. At each iteration, the evaluation of the cost function requires n simulations. Another advantage of this strategy is that, unlike C1 and C2, it can cope with action spaces of moderate dimensionality. We only evaluate with 2D spaces to make the data acquisition from real robot measurements practical.

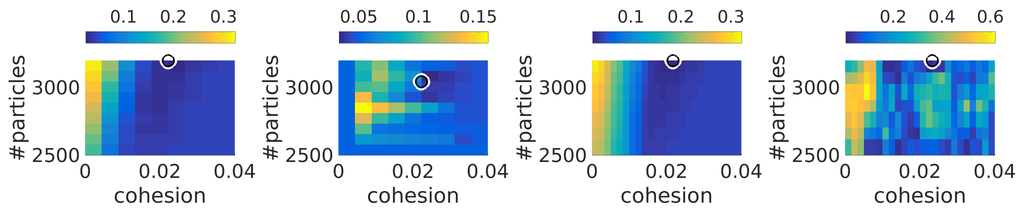
C4. Bayesian optimisation using continuous action sampling Since we observed c_r to be somewhat smooth for our choice of actions (see Section 4) we first performed Gaussian Process regression using $(\mathbf{a}_i, r_i)$ so that $\tilde{c}_r \approx c_r$ may be evaluated for any action $\mathbf{a}$. Since O_2 represents the expectation of discrepancy over all actions, at each iteration we use a primary estimate of this expectation to represent the cost. i.e. the cost function in the j^{th} iteration of the Bayesian optimizer is evaluated as $|\tilde{c}_r(\mathbf{a}_j) - c_s(\mathbf{a}_j, \theta_s)|$ with a randomly generated action $\mathbf{a}_j \sim g(\mathbf{a})$, and we used the inherent averaging properties of B.O. to effectively compute the expectation. The main advantage of this approach is that it only requires only one simulation per iteration.

3.4 Deployment: Adaptable pouring

Given the optimal simulation parameters θ_s^* from the calibration phase, the goal during deployment is to identify an action $\mathbf{a}^*$ which leads to minimal spillage when executed by the robot in conjunction with new real parameters in the scene θ_r (container shape, relative configuration, etc). We assume that θ_r is available. This optimization could either be iterative, with feedback from spillage, or one-shot. We propose and compare three techniques for the optimization during deployment.

D1. One-shot pouring (planning using simulation only) We define a loss function that is the number of poured particles that fall outside the receiving container in simulation. Using this, and a random initialization action, we perform standard Bayesian optimization to find the optimal action. At each iteration the simulator is executed once. When the reduction in the loss function between successive iterations is smaller than a threshold, we stop the optimizer and report the action identified in the last iteration as $\mathbf{a}^*$. The robot then executes the action $\mathbf{a}^*$ once to carry out the pouring task. This approach is suitable for applications where the robot has only one chance to pour the liquid. It does not require any sensors to measure the spillage after execution.

D2. Iterative pouring (planning using real execution only) In situations where the robot performs repeated pouring tasks, for example in an assembly pipeline or drinks at a bar, there is scope for



(a) Dense grid (b) Stratified sampling (c) B.O. discrete (d) B.O. continuous

Figure 4: A visualization of the normalized discrepancy (color map values) for different simulation parameters to NVIDIA FleX: Number of particles (X axis) and cohesion (Y axis). (a)-(d) are C1-C4.

iterative improvement using feedback after each pour. A simple way of achieving this is via the use of a black box Bayesian optimizer with a zero-mean Gaussian Process as the prior surrogate function. The loss function to be minimized is the weight of spilled liquid, which can be measured after every pouring event. Starting with a random action, the optimizer improves the action after every iteration thus reducing spillage on successive pours. We use this method as a baseline since it does not use simulation to improve the performance of the robot at pouring.

D3. Planned iterative pouring (planning using simulation and real execution) Finally, we propose a hybrid approach where simulation is used to guide actions executed by the robot, but with iterative improvements after each glass being poured. We replace the zero-mean surrogate in D2. with a surrogate that evaluates the averaged normalized spillage c_s over a predefined number of m simulations. After the pour is completed by the robot, as with D2, the spillage is weighed and normalized. The normalized spillage is supplied to the Bayesian optimizer as the loss function evaluated at that iteration. Each iteration requires m simulations, one pouring execution by the robot and a weighing step (of the spilled liquid).

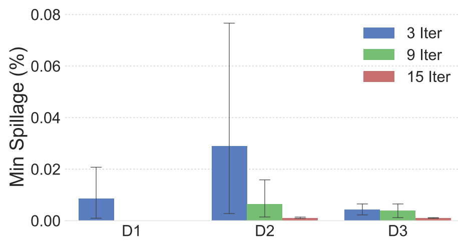
4 Experiments, results and discussion

We used a 6-DoF Universal Robots UR10 platform with a Robotiq 2-finger electric gripper to pour water. The pourer was manually placed in the gripper at mid-height. The positioning of the pourer is not important as long as it is known and consistent with the setup in simulation. The receiver was placed on a fixed tray and remained fixed while pouring. We used a Kern 2.5k weighing scale to measure the amount of spillage after each pouring experiment. Each real execution takes about 90s of which the robot only spends about 2-5s to pour. The majority of the time is spent on manually setting up the robot with 200g of water on a pourer, measuring the spillage and cleaning it up. We performed a total of about 500 real executions (total time about 15 person hours of measurements).

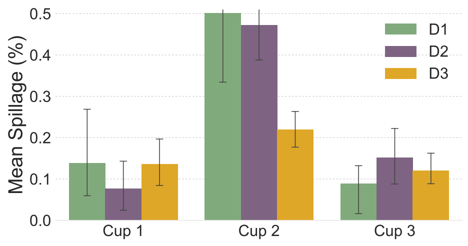
4.1 Experiments

Calibration We chose 2D action and parameter spaces. The action space consisted of angular velocity $\mathbf{a}^{(0)} = \alpha/s$ and maximum rotation (final angle) $\mathbf{a}^{(1)} = \alpha_{max}$. The parameter space consisted of $\theta_s^{(0)} = N_p$, the number of particles, and $\theta_s^{(1)} = \gamma$ the cohesion parameter in the NVIDIA FleX simulation package. We performed a total of $n = 32$ actions obtained by stratifying the action space into a 4×4 grid with two samples in each stratum. The observed data during calibration is 32 pairs of actions (performed by the robot) and normalized spillage (manually weighed and recorded) $\mathcal{D}_c = \{(\mathbf{a}_i, r_i)\}_{i=1}^{32}$. Of these, we used 28 data points to obtain optimal parameters. We then used the remaining 4 data points to assess the quality of the optimal θ_s^* obtained using different methods C1-C4 (fig. 3b). For the dense grid (C1 in sec. 3.3), we used a 10×10 grid, leading to a total number of $10 \times 10 \times 28 = 2800$ simulation calls (approx 20s each). For stratified sampling we used a 3×3 grid with two samples per grid, leading to a total number of $3 \times 3 \times 2 \times 28 = 504$ simulation calls. For C3, we ran the Bayesian optimizer for 59 iterations which led to a total number of $59 \times 28 = 1652$ calls to the simulator. Finally, for C4, we ran the optimizer to 108 iterations resulting in 108 calls to the simulator. We tracked the best normalized discrepancy over increasing numbers of simulation calls for the on-line strategies C3 and C4, to understand their path to the optimum (fig. 3c). We also visualized heatmaps of the normalized discrepancy over the parameter space (fig. 4). For methods C1 and C2 the maps are averaged over the discrepancy values for the 28 training actions. For methods C3 and C4 the maps are the posterior means of the G.P. at the last iteration of the B.O.

Deployment We chose a different, spatio-angular, action space (calibration was purely angular) for deployment. We substituted the maximum angle with $\mathbf{a}^{(1)} = x$ mm which is the relative displacement between the centers of the pouring and receiving containers along the x-axis. We also replaced the

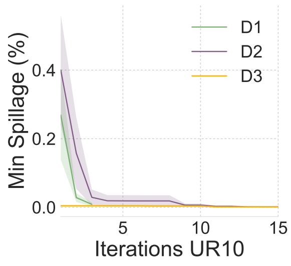


(a) deployment methods over real iterations

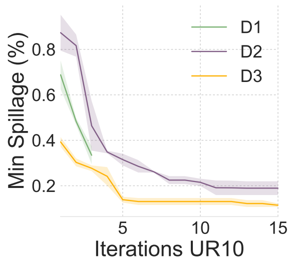


(b) adaptability to novel cups in deployment

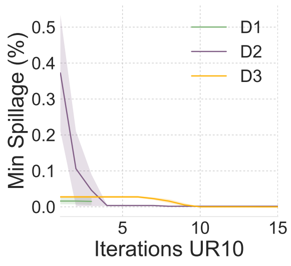
Figure 5: (a) Minimum spillage as a percentage plotted after different numbers of real iterations for Cup 1 as receiver. (b) Spillage averaged across all experiments for each of the cups. D3 performs best for the challenging case (Cup 2) while D2 performs best for the simple case (Cup 1). Error bars represent a 95% confidence interval around the mean estimate.



(a) Cup 1



(b) Cup 2



(c) Cup 3

Figure 6: Spillage plotted across real iterations (on the UR10) for the cups 1-3 and methods D1-3.

receiver used in calibration with three different containers (Cups 1-3) of different shapes and different sized openings (fig. 1). We recorded 3 independent repetitions of spillages resulting from the robot executing strategies D1, D2 and D3 (see sec. 3.4) on each receiver. We used 3, 9 and 15 iterations for the Bayesian optimization using the simulator for method D1 and 37 for method D3. We used the Upper Confidence Bound acquisition function with a constant $\kappa = 5$ that balanced the trade-off between exploration and exploitation. We performed 15 real iterative executions using the UR10 robotic arm and plotted the minimum spillage achieved upto iterations 3, 9 and 15 averaged over repetitions for the first cup (fig. 5a). We also plotted the mean spillage per real iteration achieved for each cup across the three deployment methods (fig. 5b). To understand how the strategies improved over real iterations, we plotted the spillage achieved over real iterations (averaged over repetitions) for all cups and methods in log-log scale (fig. 6).

4.2 Discussion

Calibration C4 works most efficiently for calibration (see fig. 3c) while C2 results in the least uncertainty of the optimized simulation parameters (see fig. 3b) at the cost of a small bias. C3 strikes a balance between the uncertainty of C2 and the bias of C3. We did not observe much variation in the spillage across the optimal simulation parameters produced by these methods, so we used C4 in all our tests since it requires the fewest number of simulation calls.

Deployment We observed that increasing the number of simulations is the most efficient way to reduce spillage over successive iterations. However, since the costs associated with performing one simulation and one real experiment are different we extrapolated our data to assess the space of solutions (see fig. 7). The flat areas in the curve are when B.O. "explores" the action space resulting in actions that are not improvements and the drops correspond to "exploitation" iterations. The plots confirm the intuition that spillage diminishes more rapidly over total time when the simulation is fast (fig. 7a). If simulations can be made more efficient (parallelized since they are independent), then D1 is a good choice for deployment (up to model mismatch). However, the plots also show that despite converging rapidly, D3 is better than D1 for realistic situations (fig. 7e).

Adaptability We tested adaptability by modifying the action space (from what was used in training) for deployment and by testing on 3 novel cups (that were not used in training). We observed that D2 resulted in the least spillage, on average, for simple cases (Cup 2 in fig. 5) and D3 resulted in the

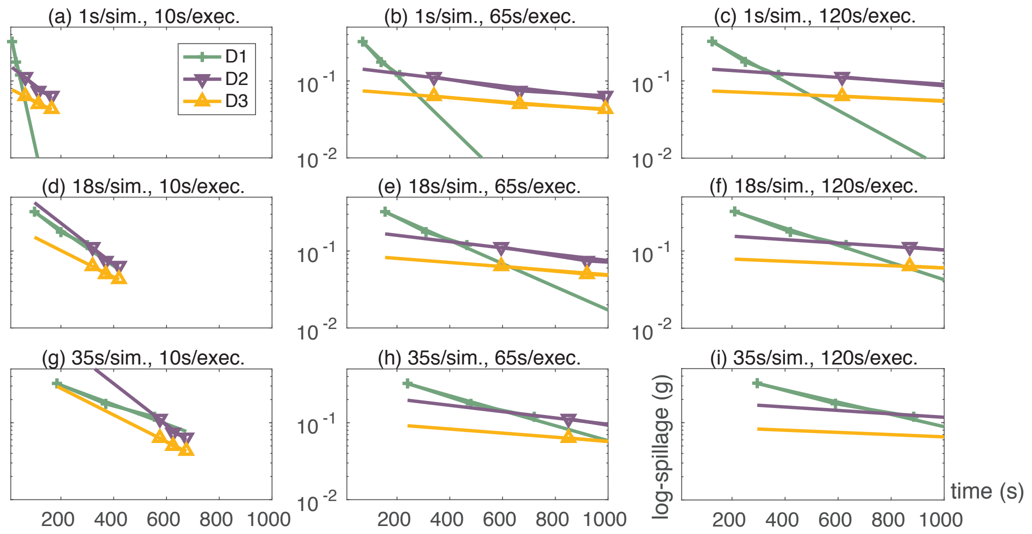


Figure 7: Comparison of log-spillage (Y-axis) against hypothesized total optimization time (X-axis) for D1 (red), D2 (green) and D3 (blue). We observed measurement error for different numbers of real and simulated iterations. Then we extrapolated the total time based on different values for times per simulation (increasing over rows) and times per real iterations (increasing left to right).

least spillage for challenging cases (Cup 2 in fig. 5). For other cases (such as Cup 3) the differences were not statistically significant. if all else is equal, D1 is preferable since it does not require multiple real iterations (hence no cleanup or sensing of the spillage is required).

Choice of action space Our method extends to action spaces of moderate dimensionality. We chose 2D action spaces because of a) the large time taken to run multiple repetitions of experiments and b) ease of visualization of the relevant costs and posterior means. Another advantage is that we do not require the action spaces for calibration and deployment to be the same. The robot could learn to accommodate for model mismatch in training and be deployed to do unforeseen tasks during calibration. We illustrated this by replacing the “final angle” action dimension with “relative position” during deployment. Without extra specifications, this caused the robot to explore a spatio-angular action space in deployment even though it was calibrated with only an angular action (fig. 3a).

Choice of parameter space As with action spaces, although our method is general enough to choose parameter spaces (for the simulator) of moderate dimensions, we only present 2D parameter spaces in this paper. We tried different combinations of parameters to NVIDIA FleX such as viscosity, cohesion, number of particles, etc. Finally, we choose cohesion and number of particles as they were sufficient to capture the behaviour of water. As can be seen in the discrepancy maps over the parameter space, the calibration algorithm has learned that the dependency along one of the dimensions is not particularly important (fig. 4). Thus, we could choose all the parameters of FleX and allow calibration to identify the important ones. As with the choice of our action space, we limited the parameter space to 2D to limit the number of manual experiments necessary.

Relationship with Dyna-Q Our proposed architecture bears a resemblance to the Dyna architecture for integrating learning, planning and reacting [33]. In that approach, the learning agent interleaves updates to a value function (in a reinforcement learning framework) based on trials in the real physical world, with updates that are driven by a loop of simulations with an internal model maintained by the agent. Whereas the typical learning rules with which Dyna has been applied, e.g., Q-learning, may not be sufficiently sample efficient for our application, we have shown that the principle is nonetheless useful when combined with alternate Bayesian Optimization based policy learning.

5 Limitations, conclusion and future work

We proposed a two-stage process for teaching robots to pour liquids based on approximate simulation. In a first stage, we optimized simulation parameters for the liquid under consideration. The shapes of containers as well as the space of actions performed during deployment may be arbitrarily chosen so long as they are known or they can be sensed. We assume that the actions are applied constantly until all the liquid is poured and that the properties of the liquid do not change between calibration and deployment. In future work, we plan to overcome the former by making the actions functions of time.

Acknowledgments

This work is supported by the Engineering and Physical Sciences Research Council (EPSRC), as part of the CDT in Robotics and Autonomous Systems at Heriot-Watt University and The University of Edinburgh. Kartic Subr is supported by a Royal Society University Research Fellowship. Tatiana López is funded by a James-Watt scholarship from the School of Mathematical and Computer Sciences at Heriot-Watt. The authors would like to thank NVIDIA for the donation of the GeForce Titan X graphics card. Marie Brown and Sam Scott-Askin for their help with the data collection and Daniel Angelov, Svetlin Penkov, Alejandro Bordallo, Joshua Smith and João Moura for the helpful discussions and support. Finally, we would like to thank the anonymous reviewers for their suggestions on improving the quality of the paper.

References

[1] Z. Pan and D. Manocha. Motion planning for fluid manipulation using simplified dynamics. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4224–4231. IEEE, 2016.

[2] C. Schenck and D. Fox. Visual closed-loop control for pouring liquids. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2629–2636. IEEE, 2017.

[3] C. Schenck and D. Fox. Reasoning about liquids via closed-loop simulation. In *Robotics: Science and Systems (RSS)*, 2017.

[4] A. Yamaguchi and C. G. Atkeson. Stereo vision of liquid and particle flow for robot pouring. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pages 1173–1180. IEEE, 2016.

[5] Z. Pan, C. Park, and D. Manocha. Robot motion planning for pouring liquids. In *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, pages 518–526, 2016.

[6] C. Pozrikidis. *Fluid dynamics: theory, computation, and numerical simulation*. Springer, 2016.

[7] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

[8] J. Stam. Real-time fluid dynamics for games. In *Proceedings of the game developer conference*, volume 18, page 25, 2003.

[9] M. Macklin and M. Müller. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32 (4):104, 2013.

[10] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4):153, 2014.

[11] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013.

[12] C. Bates, P. Battaglia, I. Yildirim, and J. B. Tenenbaum. Humans predict liquid dynamics using probabilistic simulation. In *CogSci*, 2015.

[13] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Advances in neural information processing systems*, pages 127–135, 2015.

[14] J. Kubricht, C. Jiang, Y. Zhu, S.-C. Zhu, D. Terzopoulos, and H. Lu. Probabilistic simulation predicts human performance on viscous fluid-pouring problem. In *Proceedings of the 38th annual conference of the cognitive science society*, 2016.

[15] J. Wu, J. J. Lim, H. Zhang, J. B. Tenenbaum, and W. T. Freeman. Physics 101: Learning physical object properties from unlabeled videos. In *BMVC*, 2016.

[16] A. Lerer, S. Gross, and R. Fergus. Learning physical intuition of block towers by example. *arXiv preprint arXiv:1603.01312*, 2016.

[17] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016.

[18] L. Pinto, D. Gandhi, Y. Han, Y. Park, and A. Gupta. The curious robot: Learning visual representations via physical interactions. *CoRR*, abs/1604.01360, 2016.

[19] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.

[20] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[21] A. B. Farimani, J. Gomes, and V. S. Pande. Deep learning the physics of transport phenomena. *arXiv preprint arXiv:1709.02432*, 2017.

[22] H. Wang, M. Liao, Q. Zhang, R. Yang, and G. Turk. Physically guided liquid surface modeling from videos. In *ACM Transactions on Graphics (TOG)*, volume 28, page 90. ACM, 2009.

[23] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross. Data-driven fluid simulations using regression forests. *ACM Trans. Graph.*, 34(6):199:1–199:9, Oct. 2015. ISSN 0730-0301.

[24] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating Eulerian Fluid Simulation With Convolutional Networks. *ArXiv e-prints*, July 2016.

[25] J.-M. Marin, P. Pudlo, C. P. Robert, and R. Ryder. Approximate bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012. ISSN 0960-3174.

[26] J. Lintusaari, M. Gutmann, R. Dutta, S. Kaski, and J. Corander. Fundamentals and recent developments in approximate Bayesian computation. *Systematic Biology*, 66(1):e66–e82, Jan. 2017. ISSN 1063-5157.

[27] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[28] M. Gutmann and J. Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *Journal of Machine Learning Research*, 17(125):1–47, 2016.

[29] S. Brandi, O. Kroemer, and J. Peters. Generalizing pouring actions between objects using warped parameters. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 616–621. IEEE, 2014.

[30] L. Rozo, P. Jiménez, and C. Torras. Force-based robot learning of pouring skills using parametric hidden markov models. In *Robot Motion and Control (RoMoCo), 2013 9th Workshop on*, pages 227–232. IEEE, 2013.

[31] A. Yamaguchi, C. G. Atkeson, and T. Ogasawara. Pouring skills with planning and learning modeled from human demonstrations. *International Journal of Humanoid Robotics*, 12(03): 1550030, 2015.

[32] A. Yamaguchi and C. G. Atkeson. Differential dynamic programming with temporally decomposed dynamics. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 696–703. IEEE, 2015.

[33] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, 1990.