Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

# Robot Motion Planning for Pouring Liquids

**Zherong Pan, Chonhyon Park, Dinesh Manocha** *
URL: http://gamma.cs.unc.edu/FluidMotion

**Abstract**

We present a new algorithm to compute a collision-free trajectory for a robot manipulator to pour liquid from one container to the other. Our formulation uses a physical fluid model to simulate its highly deformable motion. We present a simulation guided and optimization based method to automatically compute the transferring trajectory. We use the full-featured and accurate Navier-Stokes model that provides the fine-grained information of velocity distribution inside the liquid body. Moreover, this information is used as an additional guiding energy term for the planner. Our approach includes a tight integration between the fine-grained fluid simulator, liquid transfer controller, and the optimization-based planner. We have implemented the method using a hybrid particle-mesh fluid simulator (FLIP) and demonstrated its performance on 4 benchmarks with different cup shapes and viscosity coefficients.

Figure 1: An example of a liquid transfer task performed by a 7-DOF ClamArm robot. We use an optimization-based planner that is tightly coupled with a fluid simulator and fluid transfer controller.

## 1 Introduction

Robotic manipulation of non-rigid and physically deformable objects has been an active area of research. It frequently arises in industrial and medical applications. Most prior work in this context has been related to picking flexible objects, cable placement, surgical procedural planning, folding clothes, etc. The resulting planners deal with issues related to collision-free path computation as well as reliable physics-based simulation of deformable objects.

In this paper, we address the problem of automatically pouring liquids from one container to another using robot manipulators. Pouring liquids is a special case of a fluid manipulation tasks that frequently arises in manufacturing applications, corresponding to the dispensing of adhesives, the cleaning of parts, lubricant changes, batch material handling including fluids, etc. Other applications include service robots being used for daily chores such as cooking, cleaning, or feeding at home. One of the main challenges in these applications is modeling the motion of the fluid and taking its deformable dynamics constraints into account as part of trajectory planning. In order to accurately model the fluid, we need to solve the governing nonlinear partial differential equation, which can be computationally challenging. Prior works on motion planning with fluid constraints (Davis 2008; Kuriyama, Yano, and Hamaguchi 2008) use rather simple fluid models or are based on imitation-based learning (Langsfeld et al. 2014). However, their capabilities are limited. No good solutions are known for accurately modeling and planning the tasks of fluid pouring using robots.

**Main Results:** We present a novel algorithm for optimization-based planning that takes into account constraints corresponding to pouring liquids. One of the key contributions is the tight integration between the fine-grained fluid simulator and the optimization-based planner. We use an accurate particle-based fluid representation that has been used for accurate and fine-grained modeling of fluids (Barreiro et al. 2013; Ihmsen et al. 2014). The complexity of particle-based fluids can be high and it can take minutes to simulate a single timestep. However, we design an efficient approach to minimize the number of simulation passes required.

In our approach, the optimization-based planner tries to compute a smooth and collision-free robot joint trajectory that takes into account the liquid dynamics constraints specific to the task of pouring the liquid from one container to the other using an end-effector. These constraints are formulated as additional energy terms. To resolve the nonlinearity and non-smoothness of the fluid simulation, we design a heuristic

*Zherong, Chonhyon and Dinesh are with Department of Computer Science, the University of North Carolina, {zherong,chpark,dm}@cs.unc.edu

smooth approximation to these terms, so that conventional local optimization techniques can be used. We demonstrate the application of our planner to several challenging pouring tasks. Moreover, we evaluate its performance by changing the physical parameters such as viscosity and the size of the containers.

The rest of the paper is organized as follows: after reviewing some related works in section 2, we formulate our planning problem in section 3, and present our motion planning that takes into account fluid constraints in section 4. Finally, we highlight the performance of our algorithm on complex benchmarks in section 5.

## 2 Related Work

In this section, we give a brief overview of prior work in computational fluid dynamics (CFD), trajectory planning, and motion planning with liquid constraints.

### 2.1 Computational Fluid Dynamics

CFD is widely studied in computational mathematics and related areas. A variety of simulation techniques for liquids are known in the literature (Anderson and Wendt 1995). In practice, several factors affect the choice of a simulator for a given application. The first consideration is that the liquid free-surface, i.e. the surface where the liquid meets the air, tends to be complex and undergoes frequent topology changes. As a result, one needs efficient data structures to represent it (Scardovelli and Zaleski 1999). In our approach, we represent the liquid as a set of particles, which has been widely used in coastal engineering (Barreiro et al. 2013) and computer graphics (Ihmsen et al. 2014). Another prominent representation is based on a triangulated mesh, which is used as the basis for higher-order accurate time integrators (Eymard, Gallouët, and Herbin 2000; Harlow, Welch, and others 1965). However, these methods require complex and costly volume tracking algorithms to maintain the free surface.

Based on the underlying particle representation, one needs to select the governing equation and its discretized version so that we can predicate the position and velocity of the liquid free surfaces at each time step. In our simulator, we use the time integration scheme proposed in (Zhu and Bridson 2005), which is a discrete version of the Navier-Stokes equation of first order accuracy in both temporal and spatial domains. This is a hybrid mesh-particle solver that has been widely used in computer graphics: see (Bridson and Müller-Fischer 2007) for more details on this method and other liquid simulation techniques. An alternative is to use purely particle-based methods such as the one described in (Ihmsen et al. 2014). However, the solver in (Zhu and Bridson 2005) has better overall performance as it allows a larger timestep size while maintaining stability.

### 2.2 Trajectory Planning

In earlier works of motion planning, the main goal was to compute a collision-free path from an initial configuration to a goal configuration. More recently, these techniques have been extended to handle different kinds of constraints related to the robot or the underlying environment. Random sampling-based approaches (Stilman 2007; Berenson et al. 2009) can be used to compute a trajectory that satisfies various constraints using a direct projection of configurations into the constrained space.

Optimization techniques are used to compute a trajectory that only satisfies some hard constraints (e.g., collision-free motion), but is also optimal under some specific metrics, (e.g., smoothness or length). Many techniques based on numerical optimization have been proposed in the literature (Betts 2001). Some algorithms start with a collision-free trajectory and refine or smoothen it as a post-process using optimization techniques (Brock and Khatib 2002). Other approaches are used to compute a minimum-jerk trajectory taking into account the end-effector constraints (Olabi et al. 2010; Gasparetto and Zanotto 2008; Alatartsev et al. 2014). These methods optimize the trajectory based on an appropriate function to model the jerk motion. Some recent approaches (Ratliff et al. 2009; Park, Pan, and Manocha 2012) use a numerical solver to compute a trajectory that satisfies all the constraints. These methods typically represent various constraints, e.g., smooth trajectories, as soft constraints in terms of additional penalty terms of the objective function, and use numerical solvers to compute the resulting trajectory.

### 2.3 Planning with Fluid Constraints

There is considerable work on motion planning with deformable objects. These works include techniques to track and manipulate elastic bodies such as a rubber beam, string, or cloth (Triantafyllou et al. 2015; Li et al. 2015; Lee et al. 2014; Schulman et al. 2013), but there is relatively little work on handling liquid dynamics constraints. (Davis 2008) model the liquid pouring task in an abstract and qualitative manner and (Kunze et al. 2011) present a general framework for representing high-level information using physics-based simulation, but do not model fine-grained liquid dynamics. (Kuriyama, Yano, and Hamaguchi 2008) present an algorithm for a closely related problem: spilling avoidance. In order to find a feasible trajectory, their algorithm performs a guided stochastic search. (Langsfeld et al. 2014) present a solution for the same problem using imitation learning and use a considerable number of demonstration examples.

## 3 Overview

In this section, we introduce our problem and, give an overview of our liquid simulator and the planning framework.

### 3.1 Problem Statement

Given a robot arm whose configuration space $\mathcal{C}$ has dimension D. Each configuration is represented as a vector $\mathbf{q} \in \mathcal{R}^D$. The robot can only move in $\mathcal{C}_{free} = \mathcal{C}/\mathcal{C}_{obs}$, where $\mathcal{C}_{obs}$ represents the union of configurations that are in collision with a set of static $\mathcal{C}$-obstacles: $\{\mathcal{O}_i^s | i = 1, \cdots, M\}$. We assume that all the obstacles are rigid bodies. In addition, we introduce an additional dynamic $\mathcal{C}$-obstacle $\mathcal{O}^d(t)$ which represents the

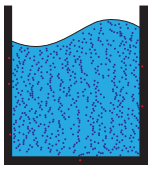

Figure 2: Particle representation of the liquid. As a given boundary condition, no liquid particle should penetrate $\mathcal{O}_i^s, \mathcal{O}^d(t)$ so the red particles are considered invalid.

source container that contains the liquid or fluid and its configuration is denoted as $\mathbf{s}(t) \in \mathcal{R}^6$. The goal of our planner is to compute a collision-free robot trajectory that transfers the liquid into the static target container $\mathcal{O}_0^s$. We assume that the robot's arm is grasping the source container at the initial configuration, so that $\mathcal{O}^d$ is treated as the end-effector. In order to represent the liquid body $\mathcal{L}$, we use a set of $L$ particles $\{p^j | j = 1, \cdots, L\}$ and store the position and velocity for each of them as illustrated in figure 2. Each configuration of the liquid body can be represented as a vector $\mathbf{p} \in \mathcal{R}^{6L}$.

Based on the above representation, we define a trajectory as a set of $N$ configurations at the discretized timesteps with a fixed interval. The trajectory of the robot arm $Q^{\mathcal{C}} \in \mathcal{R}^{DN}$ and the trajectory of the liquid $Q^{\mathcal{L}} \in \mathcal{R}^{6LN}$ over a period of time $t \in [0, T]$ are defined as:

$$Q^{\mathcal{C}} = \begin{pmatrix} \mathbf{q}_1^T & \mathbf{q}_2^T & \cdots & \mathbf{q}_N^T \end{pmatrix}^T$$
$$Q^{\mathcal{L}} = \begin{pmatrix} \mathbf{p}_1^T & \mathbf{p}_2^T & \cdots & \mathbf{p}_N^T \end{pmatrix}^T,$$

where $\mathbf{q}_i, \mathbf{p}_i$ is the configuration at time $t = \frac{i-1}{N-1}T$ so that $p_i^j$ corresponds to the $j$th particle at timestep $i$. Note that we haven't explicitly defined the trajectory of the source container $\mathcal{O}^d(t)$, since its trajectory can be computed by an end-effector transformation: $\mathbf{s}_i = T_{\mathcal{O}^d}(\mathbf{q}_i)$, which is defined by concatenating the $4 \times 4$ joint transformations from the base link.

Since liquid particles are subject to the liquid governing equation: (the Navier-Stokes equation), liquid trajectory $Q^{\mathcal{L}}$ is computed by running a fluid simulator, given $\mathcal{O}_i^s, \mathcal{O}^d(t)$ as the underlying boundary condition. For each particle set $\mathbf{p}_i$ at timestep $i$, this boundary condition states that no fluid particle can be in collision with $\mathcal{O}_i^s, \mathcal{O}^d(t)$. See figure 2. As a result, $Q^{\mathcal{L}}$ is a function of $Q^{\mathcal{C}}$ and we can define the liquid simulator $Q^{\mathcal{L}} = \mathcal{S}(Q^{\mathcal{C}})$ as:

$$Q^{\mathcal{L}} = \mathcal{S}(Q^{\mathcal{C}}) = \begin{pmatrix} \mathbf{p}_1^T & f(\mathbf{q}_1, \mathbf{p}_1)^T & \cdots & f(\mathbf{q}_{N-1}, \mathbf{p}_{N-1})^T \end{pmatrix}^T,$$

where $f(\mathbf{q}_i, \mathbf{p}_i)$ is the time-stepping function defined in algorithm 2. In our planning algorithm, $\mathcal{S}$ is treated as a black box non-smooth nonlinear function so that we can make no assumptions except evaluating $\mathcal{S}$ for a certain $Q^{\mathcal{C}}$. See the Appendix for more details on our implementation of $\mathcal{S}$.

### 3.2 Optimization-based Motion Planning

Since we want $Q^{\mathcal{C}}$ to satisfy several constraints at the same time, we formulate the motion planning problem as a continuous numerical optimization problem in the high dimensional robot trajectory space $Q^{\mathcal{C}}$ with the following objective function:

$$E(Q^{\mathcal{C}}) = c_{obs}(Q^{\mathcal{C}}) + c_{smooth}(Q^{\mathcal{C}}) + c_{fluid}(Q^{\mathcal{C}}, \mathcal{S}(Q^{\mathcal{C}})). \quad (1)$$

The first term $c_{obs}(Q^{\mathcal{C}})$ imposes the collision avoidance constraints, and many approaches from robotics can be used to compute a collision-free trajectory. In our case, the underlying optimization formulation is similar to the one used in (Kalakrishnan et al. 2011; Park, Pan, and Manocha 2012). In particular, we first compute an unsigned distance field for all static obstacles $\mathcal{O}_i^s$ so that we can perform efficient distance queries $\mathbf{d}(x)$ for any Cartesian point $x$. Next, a set of spheres $\mathcal{B}_i$ with center $x_i$ and radius $r_i$ are used to approximate the robot as well as the source container. This representation simplifies our energy formulation. Given these representations, we define:

$$c_{obs}(Q^{\mathcal{C}}) = \Sigma_{\mathcal{B}_i} \mathbf{max}(\epsilon + r_i - d(x_i), 0)\|\dot{x}_i\| +$$
$$\Sigma_{<\mathcal{B}_i, \mathcal{B}_j>} \mathbf{max}(\epsilon + r_i + r_j - \|x_i - x_j\|, 0)(\|\dot{x}_i\| + \|\dot{x}_j\|),$$

which account for both static and dynamic (self-)collision avoidance. The second term is used to impose a smoothness constraint on $Q^{\mathcal{C}}$ with Dirichlet boundary conditions:

$$c_{smooth}(Q^{\mathcal{C}}) = \frac{1}{2}\Sigma_i \|\mathbf{q}_{i+1} - \mathbf{q}_i\|^2.$$

Finally, the last term $c_{fluid}(\bullet, \mathcal{S}(\bullet))$ is an objective function that forces the fluid particles $p_i^j$ to go into the target container. We defer the formulation of $c_{fluid}$ to section 4. Since $\mathcal{S}$ is involved in the formulation of $c_{fluid}$, this term is also non-smooth and non-linear. As a result, the gradient of $c_{fluid}$ is not well defined and continuous optimization techniques cannot be used directly. Moreover, the evaluation of the function $c_{fluid}(\bullet, \mathcal{S}(\bullet))$ itself is costly. In practice, we usually have $D < 100$, while $L$ is of the order $10^{4\sim6}$. And a single call to algorithm 2 takes 3 seconds and an evaluation of $Q^{\mathcal{L}}$ with $N = 1000$ takes roughly an hour in our implementation. As a result, stochastic optimization techniques such as (Kalakrishnan et al. 2011) are not applicable because they require a considerable amount of function evaluations. Instead, we use a smooth heuristic approximation to $c_{fluid}$ so that a conventional local optimizer can be used and we ignore the gradient with respect to $\mathcal{S}$ so that the number of calls to $\mathcal{S}$ is minimized.

### 3.3 The Algorithm Overview

Our planning approach is illustrated in figure 3. The method first tries to find an initial guess $Q^{\mathcal{C}}$. It then iteratively finds a heuristic approximation $c_{fluid}^*$ to $c_{fluid}$ and computes a new $Q^{\mathcal{C}}$ by optimizing:

$$E^*(Q^{\mathcal{C}}) = c_{obs}(Q^{\mathcal{C}}) + c_{smooth}(Q^{\mathcal{C}}) + c_{fluid}^*(Q^{\mathcal{C}}),$$

until the function converges or reports failure after a fixed number of iterations. In the next section, we refine the details of how $c_{fluid}, c_{fluid}^*$ are defined.

## 4 Planning with Fluid Constraints

In this section, we present our planning algorithm, which takes into account the fluid dynamics constraints. Our

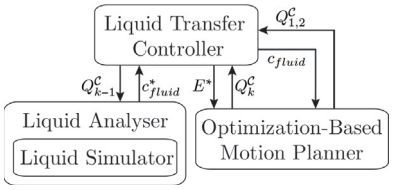


Figure 3: Overview of the various components of our motion planning algorithm for pouring liquids. The symbols used in this figure are the same as those used in algorithm 1.

formulation is based on using an energy formulation, $c_{fluid}(\bullet, \mathcal{S}(\bullet))$, and computing its approximation $c^{*}_{fluid}(\bullet)$ to accelerate the computations. Finally, we combine it with our optimization approach to compute a trajectory for the end effector and the manipulator.

## 4.1 Trajectory Initialization

The optimization algorithm needs a good initial trajectory and refines it iteratively in the high-dimensional configuration space. This boils down to computing an initial trajectory $Q^{\mathcal{C}}$ from the initial pose $\mathbf{q}_1$ following some qualitative observations of human manipulators. These observations are integrated into $c_{fluid}$ and the initial trajectory is computed by optimizing equation 1. First, for each of container $\mathcal{O}^s_0, \mathcal{O}^d$, we perform a simple shape analysis to extract the center point of the opening on top of the container $\mathbf{c}_{\mathcal{O}^s_0}, \mathbf{c}_{\mathcal{O}^d}$, which is attached with a frame, as illustrated in figure 8. We perform this computation by using a watershed algorithm (Roerdink and Meijster 2000) on the voxelized container to extract the opening and then compute the centroid. We assume that $Z = (0\,0\,1)^T$ is the negative gravity direction.

We divide the pouring task into two phases: the moving phase $P1$ where a robot orients $\mathcal{O}^d$ properly to let liquid pour out; and then the pouring phase $P2$, where the robot tries to avoid spilling by making small perturbations to the orientation of $\mathcal{O}^d$. The time periods of these two phases are given as input parameters to the algorithm. We assume $\mathbf{q}_{1\sim P}$ belongs to $P1$ and $\mathbf{q}_{P+1\sim N}$ belongs to $P2$. During $P1$, the robot moves $\mathbf{c}_{\mathcal{O}^d}$ close to $\mathbf{c}_{\mathcal{O}^s_0}$ as well as gradually turns $\mathcal{O}^d$ by an angle, (e.g., 90°), along a horizontal axis. We formulate this as an energy term:

$$c^{orient}_{fluid}(\mathbf{q}_P) = \| \begin{pmatrix} Z^T & 0 \end{pmatrix} T_{\mathcal{O}^d}(\mathbf{q}_P) \begin{pmatrix} Z \\ 0 \end{pmatrix} \|^2.$$

Note that we only apply the orientation constraint to timestep $P$ so that the robot won't move after timestep $P$, since the term $c_{smooth}$ discourages any movement. In addition, we want the opening of $\mathcal{O}^d$ to always be pointing at $\mathbf{c}_{\mathcal{O}^s_0}$. This observation introduces an additional term:

$$c^{dir}_{fluid}(\mathbf{q}_i) = \|((\mathbf{c}_{\mathcal{O}^s_0} - T_{\mathcal{O}^d}(\mathbf{q}_i)\mathbf{c}_{\mathcal{O}^d}) \times^2 (T_{\mathcal{O}^d}(\mathbf{q}_i) \begin{pmatrix} Z \\ 0 \end{pmatrix})\|^2,$$

where $a \times^2 b$ is the 2D cross product of the first $2\times1$ subvectors of $a, b$. We introduce no additional energy terms for $P2$. To get the initial trajectory, we use the following $c_{fluid}$:

$$c_{fluid}(Q^{\mathcal{C}}) = c^{orient}_{fluid}(\mathbf{q}_P) + \Sigma_{i=1\sim P}\, c^{dir}_{fluid}(\mathbf{q}_i) + c^{hint}_{fluid}, \qquad (2)$$

where the last term is a hint to guide the optimizer across highly non-linear regions:

$$c^{hint}_{fluid}(Q^{\mathcal{C}}) = w\|\mathbf{c}_{\mathcal{O}^s_0} - T_{\mathcal{O}^d}(\mathbf{q}_P)\mathbf{c}_{\mathcal{O}^d}\|^2,$$

which requires that the two opening centers are as close as possible. We set $w = 1E3$ to guide the optimizer in a first pass of optimization. We then reduce $w$ to $0.1$ as a regularization and run a second optimization. An example of the initial trajectory computed using our algorithm is shown in figure 9.

## 4.2 Simulation-Guided Trajectory Refinement

Although the initial trajectory is smooth and satisfies various qualitative criteria for the liquid pouring task, the brute-force guess by $c_{hint}(Q^{\mathcal{C}})$ may not be able to guide the fluid into $\mathcal{O}^s_0$ due to the unknown hydro-dynamic behavior of the liquid. As a result, the planning algorithm tries to adjust $Q^{\mathcal{C}}$ iteratively until convergence, which is guided by our liquid simulator.

Specifically, we first run the fluid simulator to get $Q^{\mathcal{L}} = \mathcal{S}(Q^{\mathcal{C}})$ and then replace $c_{hint}(Q^{\mathcal{C}})$ with a liquid guiding term $c^{guide}_{fluid}(Q^{\mathcal{L}})$. To define this term, we make use of the particle streamline. Each particle $p^j$ is associated with a streamline $\mathcal{P}^j = \{p^j_i | i = 1, \cdots, N\}$, as illustrated in figure 10. An intuitive requirement for successful transfer is that every $\mathcal{P}^j$ should pass through $\mathbf{c}_{\mathcal{O}^s_0}$. As a result, we have $c^{guide}_{fluid}(Q^{\mathcal{L}}) = \Sigma_j \mathbf{dist}(\mathcal{P}^j(\mathcal{S}(Q^{\mathcal{C}})), \mathbf{c}_{\mathcal{O}^s_0})$. Now the big problem is that we don't have the gradient information for $\mathcal{S}(Q^{\mathcal{C}})$, so that the optimization of $c^{guide}_{fluid}(Q^{\mathcal{L}})$ requires numerical differentiation or stochastic optimization algorithms such as (Kuriyama, Yano, and Hamaguchi 2008), which usually require a lot of costly evaluation of $\mathcal{S}$.

Instead, we use a heuristic smooth approximation to $c^{guide}_{fluid}(Q^{\mathcal{L}})$ by performing a simple analysis of $\mathcal{P}^j$. We assume that $\mathcal{P}^j$ is comprised of three stages. During the first stage $S1$, the particle lies within the container $\mathcal{O}^d$ and it is the pressure/viscosity terms that affect $\mathcal{P}^j_{S1}$. During the phase $S2$, the particle leaves the container and the gravity term dominates its motion so that $\mathcal{P}^j_{S2}$ should roughly follow a quadratic curve (this assumption is verified in section 7). Finally, during $S3$ the particle is in $\mathcal{O}^s_0$, and $\mathcal{P}^j_{S3}$ is again dominated by pressure/viscosity terms. These stages are illustrated in figure 10.

Overall the near quadratic curve $\mathcal{P}^j_{S2}$ has the most effect in $c^{guide}_{fluid}$. Indeed, $\mathbf{c}_{\mathcal{O}^s_0}$ can never pass through $\mathcal{P}^j_{S1}$ or $\mathcal{P}^j_{S3}$, because they are both within the container. Thus $\mathcal{P}^j_{S2}$ alone already provides sufficient information to adjust the position of $\mathcal{O}^d$. In order to avoid gradient evaluation, we can simply ignore the dependency of $\mathcal{P}^j_{S2}$ on $Q^{\mathcal{C}}$ to get the zeroth-order approximation of $\mathcal{S}$ to define:

$$c^{guide*}_{fluid} = \Sigma_j \mathbf{dist}(T_{\mathcal{O}^d}(\mathbf{q}_{S2})\mathcal{P}^j_{S2}, \mathbf{c}_{\mathcal{O}^s_0}),$$

where the function $\mathbf{dist}(\mathcal{P}, \mathbf{c}_{\mathcal{O}^s_0})$ returns the closest point from $\mathbf{c}_{\mathcal{O}^s_0}$ to a piecewise linear curve segment, whose gradient can be locally evaluated. Note that $\mathcal{P}^j$ is replaced with $\mathcal{P}^j_{S2}$ and it is assumed to be independent of $\mathcal{S}(Q^{\mathcal{C}})$ so that we don't need to evaluate the gradient of $\mathcal{S}$. Instead, we introduce

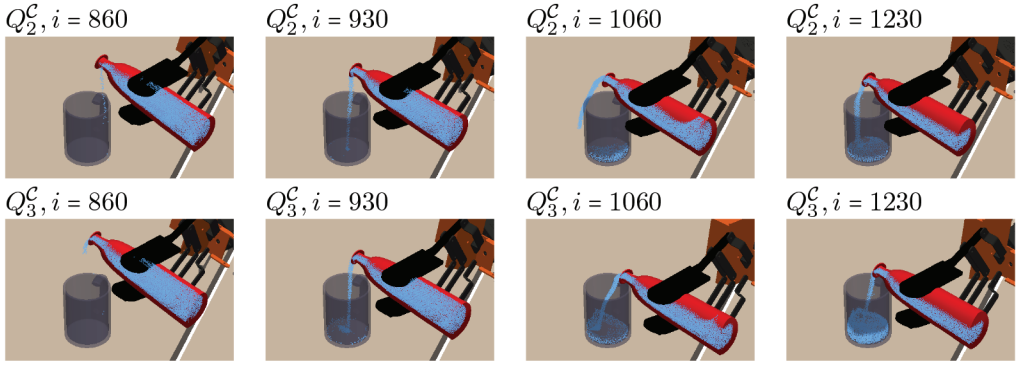


Figure 4: Different timesteps $i$ of the optimized robot trajectory $Q_k^{\mathcal{C}}$ and simulated liquid trajectory $\mathcal{S}(Q_k^{\mathcal{C}})$ after $k$ iterations of algorithm 1. We use small dynamic viscosity coefficient $\mu = 0.01$ for the liquid in this benchmark. After only one iteration, the flow becomes centered around the target opening $\mathbf{c}_{\mathcal{O}_0^s}$.

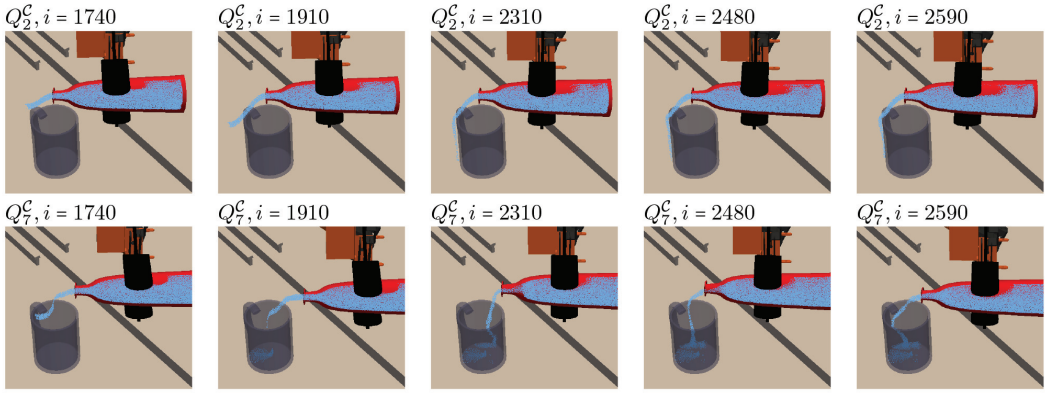


Figure 5: Different timesteps $i$ of the optimized robot trajectory $Q_k^{\mathcal{C}}$ and the simulated liquid trajectory $\mathcal{S}(Q_k^{\mathcal{C}})$ after $k$ iterations of algorithm 1. We use a large dynamic viscosity coefficient $\mu = 0.5$ for the liquid. The error reduction is slower and even after 5 iterations the flow is still not well centered around the target opening $\mathbf{c}_{\mathcal{O}_0^s}$.

a heuristic assumption that $\mathcal{P}_{S2}^j$ is rigidly attached to the container at the beginning timestep of $S2$. Here $T_{\mathcal{O}^d}(\mathbf{q}_{S2})\mathcal{P}_{S2}^j$ is defined by transforming each vertex of $\mathcal{P}_{S2}^j$ by $T_{\mathcal{O}^d}(\mathbf{q}_{S2})$. This is a reasonable assumption because we can only control the particle when it is still within the container $\mathcal{O}^d$, so that we control it at the last moment before it leaves $\mathcal{O}^d$, which is the beginning of stage $S2$. Now we can formulate $c_{fluid}$ as:

$$c^*_{fluid}(Q^{\mathcal{C}}) = c^{orient}_{fluid}(\mathbf{q}_P) + \Sigma_{i=1\sim P} c^{dir}_{fluid}(\mathbf{q}_i) + c^{guide*}_{fluid}(Q^{\mathcal{C}}). \quad (3)$$

The remaining problem is to find the timestep index corresponding to the beginning of $S2$. To do this, we make use of the fact that $\mathcal{P}_{S2}$ should approximate a quadratic curve and solve the following least square problem:

$$\mathbf{argmin}_{v_0,c_0} \Sigma_{i\epsilon\mathcal{W}} \|x_i^j - (\frac{1}{2}gt_i^2 + v_0 t_i + c_0)\|^2, \quad (4)$$

where $g$ is the gravity, $t_i = \frac{i-1}{N-1}T$, and $\mathcal{W}$ is a sliding window through all consecutive $K$ timesteps in $\mathcal{P}^j$, from which we pick the $\mathcal{W}$ with minimum error. Next, we propagate the curve by greedily expand the window $\mathcal{W}$ by one timestep until the relative error is above some threshold $\epsilon$. The beginning of $S2$ is then identified with the beginning of $\mathcal{W}$. Our overall pipeline is illustrated in algorithm 1.

## 5 Implementation and Results

In this section, we present the implementation details and highlight the performance of our trajectory planning algorithm on challenging benchmarks. We use ROS (Quigley et al. ) with a 7-DOF ClamArm as the underlying manipulator. All the optimization computations are performed using the Augmented Lagrangian algorithm, with the L-BFGS algorithm as the sub-problem solver. In terms of fluid dynamics, our implementation follows the outline of (Zhu and Bridson 2005) combined with (Batty, Bertails, and Bridson 2007) for boundary handling. We precompute a signed distance field for both $\mathcal{O}^d$ and $\mathcal{O}_0^s$ as the solid boundaries, and we assume that there is no collision between the robot and the liquid particles.

The performance of planning highly depends on the use of an appropriate mesh data-structure for liquid pressure computation. A simple implementation can be based on a uniform grid whose memory complexity is $\mathcal{O}(V/V0)$, where

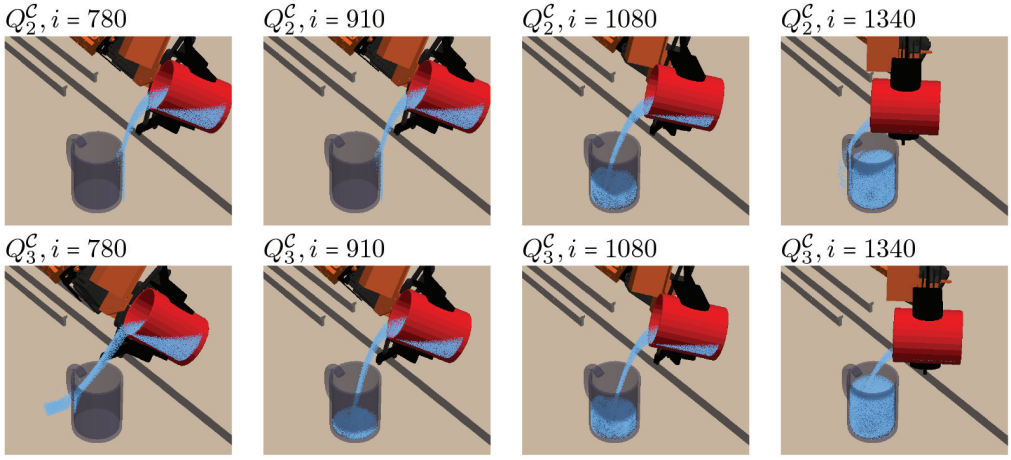


Figure 6: Different timesteps $i$ of the optimized robot trajectory $Q_k^{\mathcal{C}}$ and simulated liquid trajectory $\mathcal{S}(Q_k^{\mathcal{C}})$ after $k$ iterations of algorithm 1. We use small dynamic viscosity coefficient $\mu = 0.01$ for this liquid. Again, the flow is centered around the target opening $\mathbf{c}_{\mathcal{O}_0^s}$ but there is some splash at the beginning $i = 780$ which cannot be reduced by further iterations.

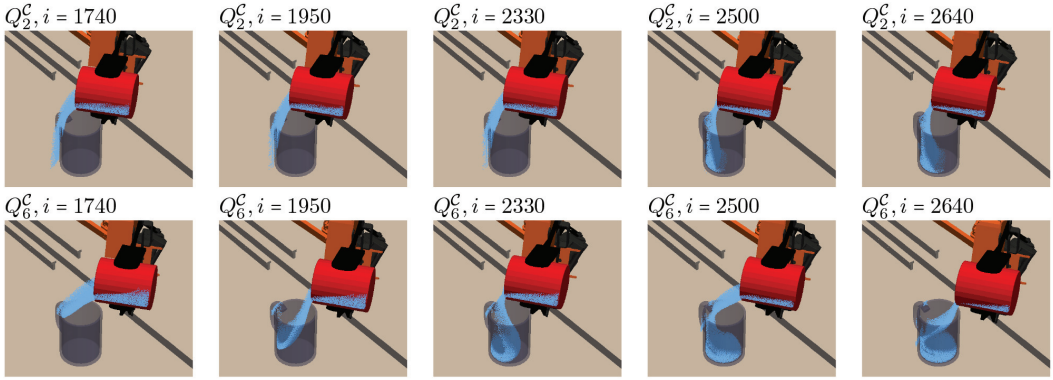


Figure 7: Different timesteps $i$ of the optimized robot trajectory $Q_k^{\mathcal{C}}$ and the simulated liquid trajectory $\mathcal{S}(Q_k^{\mathcal{C}})$ after $k$ iterations of algorithm 1. We use a large dynamic viscosity coefficient $\mu = 0.5$ for the liquid. In the same way as the case with figure 5, error reduction is slower. After 4 iterations, the robot moves $\mathcal{O}^d$ back and forth, trying to make the viscous liquid fall inside $\mathcal{O}_0^s$.

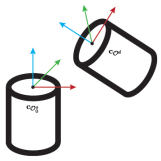

Figure 8: The local coordinate system and the center of the opening for source/target container, used to define the objective function equation 2.

$V$ is the volume of the bounding box and $V0$ is the volume of each mesh cell. However, since particle distribution is highly irregular, a uniform grid can be inefficient in terms of memory footprint. Instead, we use an adaptive data structure: the DT-grid (Nielsen and Museth 2006), which only assigns mesh cells around particles so that the complexity becomes $\mathcal{O}(L)$, where $L$ is the number of particles.

All the results were generated on a desktop computer with

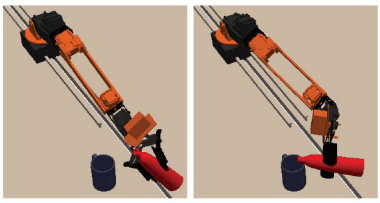

Figure 9: An example of the initial trajectory for the benchmark of figure 4 at timestep $i = 250/1000, 700/1000$, which is found using our two-stage algorithm.

an $i7 - 4790$ 8-core CPU 3.6GHz and 8GB of memory. table 1 lists the parameters used within our liquid simulator. According to the CFL condition (Bridson and Müller-Fischer 2007), there is a maximum allowable time step size as a function of particle velocity. This is not considered in our current implementation. However, no violations of CFL conditions

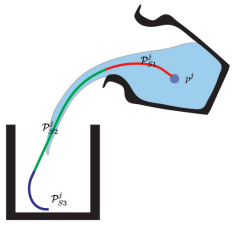

Figure 10: Three stages in the streamline of a single particle $p^j$: $S1$ in red, $S2$ in green and $S3$ in blue.

**Algorithm 1** Liquid Pouring Trajectory Optimization Algorithm

**Input:** Center of opening $\mathbf{c}_{\mathcal{O}_0^s}, \mathbf{c}_{\mathcal{O}^d}$, initial pose $\mathbf{q}_1$
**Output:** $Q^{\mathcal{C}}$ to transfer water from $\mathcal{O}^d$ to $\mathcal{O}_0^s$

▷ Run a first pass of initialization
Set $w = 1E3$ and set $\mathbf{q}_i = \mathbf{q}_1$ as initial guess
$Q_1^{\mathcal{C}} = \mathbf{argmin}\ (1)$ with (2)
▷ Run a second pass of initialization
Set $w = 0.1$ and $Q_1^{\mathcal{C}}$ as initial guess
$Q_2^{\mathcal{C}} = \mathbf{argmin}\ (1)$ with (2)
▷ Main iteration
Set $k = 3$
**while** true **do**
    Run liquid simulation $Q^{\mathcal{L}} = \mathcal{S}(Q^{\mathcal{C}})$
    ▷ Find the beginning of $S2$ for each particle $p^j$
    **for all** $j$ **do**
        Find $\mathcal{P}_{S2}^j$ by solving equation 4
    **end for**
    ▷ Update the trajectory
    Set $Q_{k-1}^{\mathcal{C}}$ as initial guess
    $Q_k^{\mathcal{C}} = \mathbf{argmin}\ (1)$ with (3)
    ▷ exit if the difference is small
    **if** $\|Q_{k-1}^{\mathcal{C}} - Q_k^{\mathcal{C}}\| < \epsilon$ **then**
        Return $Q_k^{\mathcal{C}}$
    **end if**
    Set $k = k + 1$
**end while**

are observed. The overall performance of our algorithm is shown in table 2. The underlying liquid simulator is the most expensive step in the overall computation.

We have evaluated our approach on two sets of benchmarks with different $\mathcal{O}^d$ shapes and fluid viscosity. figure 4 illustrates several timesteps of the manipulator and fluid motion trajectory. In this case, we use a small viscosity coefficient, $\mu = 0.01$. The flow becomes centered around $\mathbf{c}_{\mathcal{O}_0^s}$ after only 1 iteration of algorithm 1 and it converged after 2 iterations. We then increase the viscosity to $\mu = 0.5$, which corresponds to some sticky material such as mud or honey. In this case, the problem of motion planning with fluid constraints becomes more challenging, since the liquid simulation operator $\mathcal{S}$ plays an important role as $\mu$ increases. However, our algorithm is still able to guide liquid into $\mathcal{O}_0^s$ after six iterations, as illustrated in figure 5. Note that the flow is less centered around $\mathbf{c}_{\mathcal{O}_0^s}$ when compared with figure 4.

| Parameter | Value (Unit) |
|---|---|
| Avg. Particle Radius | $0.0075(m)$ |
| Gravity | $-9.81Z(m/s)$ |
| $\Delta t = T/N$ | $0.01(s)$ |
| No. Particles (Long Bottle) | 60000 |
| No. Particles (Cup with Handle) | 56000 |
| $T$ | $15(s)$ |
| $T_{P1}$ | $6(s)$ |

Table 1: Parameters used in our liquid simulator: Average particle radius, gravity, time step size, number of particles needed to fill the long bottle, number of particles needed to fill the cup with handle, total sequence duration, duration of P1.

| Substep | Computation Time (Unit) |
|---|---|
| One step of fluid simulation | $3.3(s)$ |
| One pass of fluid simulation | $0.9(h)$ for $N = 1000$ |
| Computing $Q_1^{\mathcal{C}}$ and $Q_2^{\mathcal{C}}$ | $< 10(s)$ |
| Computing $Q_{k>2}^{\mathcal{C}}$ | $< 3(s)$ |

Table 2: Performance of each sub-step in algorithm 1.

In the previous benchmarks, the source container $\mathcal{O}^d$ is a bottle with a small opening, so that the particle outflow velocity distribution is highly concentrated. We relax this assumption in the next benchmark shown in figure 6, where we use the handled cup as $\mathcal{O}^d$ to allow a wider opening. Our iterative algorithm works well and can reduce the error in a few iterations. Again we then run the algorithm on a highly viscous fluid with $\mu = 0.5$. The result is shown in figure 7. As was the case in figure 5, more iterations are needed. After five iterations, the robot does some back and forth adjustment to make the viscous sheet fall in the cup.

## 6 Conclusions, Limitations and Future Work

In this paper, we present a novel algorithm for trajectory planning for pouring liquids. Our formulation takes into account collision-free, smoothness, and dynamics constraints of the robot combined with fluid simulation constraints. In particular, we use a fine-grained liquid simulator to guide the trajectory optimization for the robot manipulator and integrate them into an optimization-based motion planning framework. Given the non-smoothness and nonlinearity of the liquid simulation procedure, we use a heuristic approximation to guide the robot manipulator towards the desired goal configuration. We have demonstrated our approach with different container shapes and the physical parameters of the fluid (e.g., viscosity).

Our approach has several limitations. Our current planner and liquid transfer controller are specifically designed for pouring tasks. We also assume that the environment is static and the target container is fixed and oriented in an upright configuration. We also assume that the liquid model is the single-phase incompressible Navier-Stokes model and ignore the air pressure surrounding fluid particles.The energy-based approximation considerably speeds up the computation, but

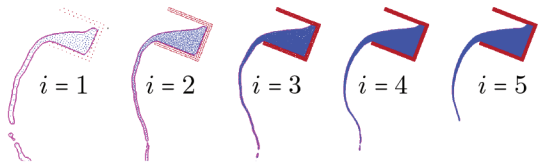


Figure 11: We ran 5 passes of 2D fluid simulations with the same initial setting: pouring a cup of water. For each larger $i$, we double the resolution and halve the particle radius. A mesh is reconstructed for each timestep in each simulation (purple).

may not always find a good trajectory that can satisfy all the constraints, including fluid dynamics. The complexity of the approach increases with the number of particles used to approximate the fluid, and we use low resolution particle approximation for efficiency reasons.

There are many avenues for future work. Besides overcoming these limitations, we would like to evaluate our planner in complex scenarios and finally test its performance on real robot hardware. It would be useful to perform other fluid manipulation tasks and design appropriate motion planning strategies. We would like to accelerate the computations using parallel algorithms that exploit multi-core CPUs and many-core GPUs.

## 7 Appendix: Liquid Simulator

The liquid simulator $\mathcal{S}$ used in our planner proceeds by repeatedly applying the time integrator: $\mathbf{p}_{i+1} = f(\mathbf{q}_i, \mathbf{p}_i)$. The governing equation of our solver is the single-phase incompressible Navier-Stokes equation:

$$\frac{Du}{Dt} = \mu\nabla\cdot(\nabla u + \nabla u^T) + g - \nabla p$$
$$\nabla\cdot u = 0, \tag{5}$$

where $u$ is the velocity field inside the liquid body. On the right hand side are three body force terms that drive the liquid particles: the isotropic viscosity force with coefficient $\mu$, the gravity force $g$, and the pressure force $p$. The dynamic viscosity coefficient $\mu$ describes how sticky the liquid is. For example, oil and honey have a larger $\mu$ compared to water. Moreover, we introduce additional constraints $\nabla\cdot u = 0$, which essentially imply that the volume of fluid is conserved, so that the unknown pressure can be identified as the Lagrangian multiplier and the resulting system is closed.

There are many ways to discretize equation 5; our simulator is based on (Zhu and Bridson 2005), and it is described in algorithm 2. This is essentially a time-splitting integrator that accounts for each of the three terms separately. In this formulation, $\nabla\cdot(\nabla u + \nabla u^T)(x)$, is the evaluation of the differential operator $\nabla\cdot(\nabla u + \nabla u^T)$ at $x$ and $\nabla\cdot u(x)$ is the evaluation of $\nabla\cdot u$ at $x$. In order to perform these evaluations, we use a background grid and a finite difference scheme. These two terms require solving a globally coupled linear system, which becomes the computational bottleneck of the resulted simulator. For more details, we refer the readers to (Bridson and Müller-Fischer 2007).

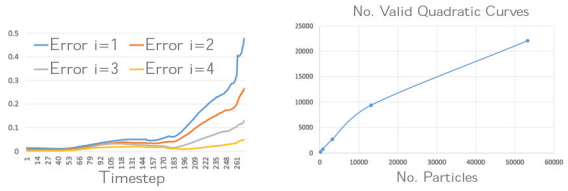


Figure 12: Left: Error plot for $\mathbf{mesh}(i,t), i = 1,\cdots,4$ with respect to timestep $t$. As $i$ increases, the error is approximately halved. Right: Number of particles whose streamline can be approximated by a quadratic curve with error less than $\epsilon$, plotted against the total number of particles used in simulation. We choose $\epsilon$ equal to the particle radius in all the experiments.

**Algorithm 2** The time integrator $\mathbf{p}_{i+1} = f(\mathbf{q}_i, \mathbf{p}_i)$

**Input:** All particle $p_i^j$'s position $x_i^j$ and velocity $u_i^j$
**Output:** New positions $x_{i+1}^j$ and velocities $u_{i+1}^j$

▷ Apply gravity force
**for all** $j$ **do**
  $u_i^{j*} = u_i^j + g\Delta t$
**end for**
▷ Apply viscosity force
**for all** $j$ **do**
  $(u_i^{j**} - u_i^{j*}) = \Delta t\mu\nabla\cdot(\nabla u^{**} + \nabla u^{**^T})(x_i^j)$
**end for**
▷ Apply pressure force
**for all** $j$ **do**
  $(u_{i+1}^j - u_i^{j**}) = -\Delta t\nabla p^j,\ \nabla\cdot u(x_i^j) = 0$
**end for**
▷ Particle position update
**for all** $j$ **do**
  $x_{i+1}^j = x_i^j + u_{i+1}^j\Delta t$
**end for**

To work with a motion planner, it is important for this fluid simulator to be convergent. Here we briefly verify the first order spatial-temporal convergence property of our fluid simulator. To do this, we perform 5 passes of 2D simulations of pouring a cup of water as illustrated in figure 11, under different resolutions. For each timestep in each simulation, we reconstruct a mesh to get 5 sequences of meshes denoted as $\mathbf{mesh}(i,t)$, where $i$ is the sequence id and $t$ is the timestep id. To investigate the convergence property, we treat $i = 5$ as the groundtruth simulation and define the error metric as: $\mathbf{Error}(\mathbf{mesh}(i,t)) = d_m(\mathbf{mesh}(i,t), \mathbf{mesh}(5,t))$, where $d_m$ is the mean error defined in (Aspert, Santa Cruz, and Ebrahimi 2002). With this definition, the error plot for $\mathbf{mesh}(i,t), i = 1,\cdots,4$ is illustrated in figure 12, where it is obvious that the mean error is approximately halved as we double the resolution.

In addition, our objective function equation 4 is based on the assumption that particle streamlines can be approximated as quadratic curves. For each of the 5 simulation passes, we can verify this assumption by counting the number of parti-

cles whose streamline can be approximated by a quadratic curve with an error smaller than $\epsilon$, and plot it against the total number of particles used in simulation. From the resulting figure 12, we are confident that it is safe to make such an assumption for at least half of the particles.

## Acknowledgement

This research is supported in part by ARO Contract W911NF-14-1-0437, NSF award 1305286.

## References

Alatartsev, S.; Belov, A.; Nykolaichuk, M.; and Ortmeier, F. 2014. Robot Trajectory Optimization for the Relaxed End-Effector Path. In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*.

Anderson, J. D., and Wendt, J. 1995. *Computational fluid dynamics*, volume 206. Springer.

Aspert, N.; Santa Cruz, D.; and Ebrahimi, T. 2002. Mesh: measuring errors between surfaces using the hausdorff distance. In *ICME (1)*, 705–708.

Barreiro, A.; Crespo, A.; Domínguez, J.; and Gómez-Gesteira, M. 2013. Smoothed particle hydrodynamics for coastal engineering problems. *Computers & Structures* 120:96–106.

Batty, C.; Bertails, F.; and Bridson, R. 2007. A fast variational framework for accurate solid-fluid coupling. In *ACM Transactions on Graphics (TOG)*, volume 26, 100. ACM.

Berenson, D.; Srinivasa, S. S.; Ferguson, D.; Collet, A.; and Kuffner, J. J. 2009. Manipulation planning with workspace goal regions. In *Proceedings of IEEE International Conference on Robotics and Automation*, 618–624.

Betts, J. T. 2001. Practical methods for optimal control and estimation using nonlinear programming. In *Advances in design and control*, volume 3. Siam.

Bridson, R., and Müller-Fischer, M. 2007. Fluid simulation: Siggraph 2007 course notes video files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 courses*, 1–81. ACM.

Brock, O., and Khatib, O. 2002. Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research* 21(12):1031–1052.

Davis, E. 2008. Pouring liquids: A study in commonsense physical reasoning. *Artificial Intelligence* 172(12):1540–1578.

Eymard, R.; Gallouët, T.; and Herbin, R. 2000. Finite volume methods. *Handbook of numerical analysis* 7:713–1018.

Gasparetto, A., and Zanotto, V. 2008. A technique for time-jerk optimal planning of robot trajectories. *Robotics and Computer-Integrated Manufacturing* 24(3):415–426.

Harlow, F. H.; Welch, J. E.; et al. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of fluids* 8(12):2182.

Ihmsen, M.; Orthmann, J.; Solenthaler, B.; Kolb, A.; and Teschner, M. 2014. Sph fluids in computer graphics.

Kalakrishnan, M.; Chitta, S.; Theodorou, E.; Pastor, P.; and Schaal, S. 2011. STOMP: Stochastic trajectory optimization for motion planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, 4569–4574.

Kunze, L.; Dolha, M. E.; Guzman, E.; and Beetz, M. 2011. Simulation-based temporal projection of everyday robot object manipulation. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 107–114. International Foundation for Autonomous Agents and Multiagent Systems.

Kuriyama, Y.; Yano, K.; and Hamaguchi, M. 2008. Trajectory planning for meal assist robot considering spilling avoidance. In *Control Applications, 2008. CCA 2008. IEEE International Conference on*, 1220–1225. IEEE.

Langsfeld, J. D.; Kaipa, K. N.; Gentili, R. J.; Reggia, J. A.; and Gupta, S. K. 2014. Incorporating failure-to-success transitions in imitation learning for a dynamic pouring task.

Lee, A. X.; Huang, S. H.; Hadfield-Menell, D.; Tzeng, E.; and Abbeel, P. 2014. Unifying scene registration and trajectory optimization for learning from demonstrations with application to manipulation of deformable objects. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 4402–4407. IEEE.

Li, Y.; Yue, Y.; Xu, D.; Grinspun, E.; and Allen, P. K. 2015. Folding deformable objects using predictive simulation and trajectory optimization. In *Proceedings of the 27th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Nielsen, M. B., and Museth, K. 2006. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing* 26(3):261–299.

Olabi, A.; Béarée, R.; Gibaru, O.; and Damak, M. 2010. Feedrate planning for machining with industrial six-axis robots. *Control Engineering Practice* 18(5):471–482.

Park, C.; Pan, J.; and Manocha, D. 2012. ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In *Proceedings of International Conference on Automated Planning and Scheduling*.

Quigley, M.; Faust, J.; Foote, T.; and Leibs, J. Ros: an open-source robot operating system.

Ratliff, N.; Zucker, M.; Bagnell, J. A. D.; and Srinivasa, S. 2009. CHOMP: Gradient optimization techniques for efficient motion planning. In *Proceedings of International Conference on Robotics and Automation*, 489–494.

Roerdink, J. B., and Meijster, A. 2000. The watershed transform: Definitions, algorithms and parallelization strategies.

Scardovelli, R., and Zaleski, S. 1999. Direct numerical simulation of free-surface and interfacial flow. *Annual review of fluid mechanics* 31(1):567–603.

Schulman, J.; Lee, A.; Ho, J.; and Abbeel, P. 2013. Tracking deformable objects with point clouds. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 1130–1137. IEEE.

Stilman, M. 2007. Task constrained motion planning in robot joint space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3074–3081.

Triantafyllou, D.; Mariolis, I.; Kargakos, A.; Malassiotis, S.; and Aspragathos, N. 2015. A geometric approach to robotic unfolding of garments. *Robotics and Autonomous Systems*.

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