Learning compositional models of robot skills for task and motion planning

Zi Wang[*,1,2]**, Caelan Reed Garrett**[*,1]**, Leslie Pack Kaelbling**[1]**, and Tomás Lozano-Pérez**[1]

Abstract

The objective of this work is to augment the basic abilities of a robot by learning to use sensorimotor primitives to solve complex long-horizon manipulation problems. This requires flexible generative planning that can combine primitive abilities in novel combinations and thus generalize across a wide variety of problems. In order to plan with primitive actions, we must have models of the actions: under what circumstances will executing this primitive successfully achieve some particular effect in the world?
We use, and develop novel improvements on, state-of-the-art methods for active learning and sampling. We use Gaussian process methods for learning the constraints on skill effectiveness from small numbers of expensive-to-collect training examples. Additionally, we develop efficient adaptive sampling methods for generating a comprehensive and diverse sequence of continuous candidate control parameter values (such as pouring waypoints for a cup) during planning. These values become end-effector goals for traditional motion planners that then solve for a full robot motion that performs the skill. By using learning and planning methods in conjunction, we take advantage of the strengths of each and plan for a wide variety of complex dynamic manipulation tasks. We demonstrate our approach in an integrated system, combining traditional robotics primitives with our newly learned models using an efficient robot task and motion planner. We evaluate our approach both in simulation and in the real world through measuring the quality of the selected primitive actions. Finally, we apply our integrated system to a variety of long-horizon simulated and real-world manipulation problems.

Keywords

Machine Learning, Active Learning, Task and Motion Planning, Gaussian Process, Manipulation

1 Introduction

For robots to be useful in a home environment, they will have to be endowed with a foundational set of capabilities, such as locomotion and basic object manipulation. They will then have to build on those capabilities by acquiring more specialized skills such as pouring milk or scooping cereal. It is critical that these skills be acquired *efficiently*, with relatively few training examples, and that they be used *compositionally*, combining with existing skills to generalize to a wide variety of situations and purposes for which that skill can be usefully deployed.

The vast majority of research in robot learning has focused on acquiring closed-loop sensorimotor skills, ranging from pouring (Yamaguchi et al. 2014) to manipulating a Rubik's cube (OpenAI et al. 2019). Very little work has focused on how to actually combine and execute these skills to address problems in the world (but see the work of Wang and Kroemer (2019) for a good example). In this paper, we provide a framework for integrating new skills with existing ones by learning *skill models* and using them to plan sequences of skill executions to achieve long-horizon goals in complex environments.

The overall class of problems we wish to address, known as *task and motion planning* (TAMP), considers a robot carrying out tasks in an environment such as a kitchen, storage depot, and construction site. These tasks require the robot to manipulate multiple objects, potentially moving

Figure 1. Making coffee in *KitchenPR2*, which requires pouring, scooping, dumping, and stirring.

things out of the way, or putting them into or taking them out of containers, as well as to perform additional operations, such as pouring or cleaning, in service of a high-level objective. TAMP planners combine robot motion planning

[*]Equal contribution. [1]MIT CSAIL, MA. [2]Now at Google.

Corresponding author:
Caelan Reed Garrett, MIT CSAIL, 32 Vassar St, Cambridge, MA 02139.
Email: caelan@csail.mit.edu

with the selection of continuous parameters, for example governing grasps and placements of objects, and the high-level selection of which operations to perform on which objects in what order.

In this paper, we will use making coffee as a motivating example. This task involves picking and placing a variety of objects, such as cups and spoons. It also involves pouring cream from one container to another and scooping sugar from a bowl into a cup. These actions need to be performed in a wide variety of object arrangements on a tabletop, with the relevant objects in arbitrary initial poses and possibly in the presence of extraneous objects. The robot should be able to apply its skills of picking, placing, and pushing objects as well as its learned skills of pouring and scooping to enable successful completion of the coffee-making task in these arbitrary tabletop environments.

Figure 1 demonstrates this task using a real-world PR2 robot. Figure 5 (*right*) shows a 3D simulation version of this task in PyBullet (Coumans and Bai 2016–2019). We use simulations to carry out extensive evaluation of our learning algorithms. However, crucially, learning on the real robot *does not* rely on the simulation. We do not want to be limited to skills for which a high-fidelity simulation is required, so we attempt to learn on a real robot in as few trials as possible.

TAMP problems are *hybrid*, requiring discrete and continuous choices. TAMP planning approaches generally combine aspects of discrete planning methods from symbolic artificial intelligence (AI) with constrained sampling or direct optimization to select continuous parameters. Our approach will be to learn models of new skills that allow them to be incorporated into a TAMP framework and immediately combined with existing skills in service of achieving high-level goals.

Given a parameterized sensorimotor policy (a *skill*) $\pi_{O(\omega)}$ that was intended to achieve some condition in the world (such as liquid being in a particular bowl or spoon), our goal will be to characterize it in a form that enables a TAMP planner to deploy it in combination with existing skills. To do this, we need to formally describe the intended effects of the new skill as well as conditions on the state in which the skill is initiated that would guarantee that the intended effect occurs. For example, the intended effect of a pouring skill is that liquid be in some destination container, and the precondition of that effect is that the robot is holding some other container that has liquid in it.

There are three important constraints on the process of learning the preconditions and effects of a skill:

1. Learning should characterize a *comprehensive* set of control parameter values instead of a single value. The predicted values are subjected to downstream constraints, for example, arising from robot kinematics and collision avoidance. There might not be any robot motion that satisfies these for an individual control value.

2. The result of learning should have *quantified uncertainty*: that is, we should be able to characterize possible starting states for skill execution in terms of how sure we are that the intended effect will occur. Knowing this will allow the TAMP planner, as far as possible given the problem-solving context, to use the skill in a way that it is confident will succeed.
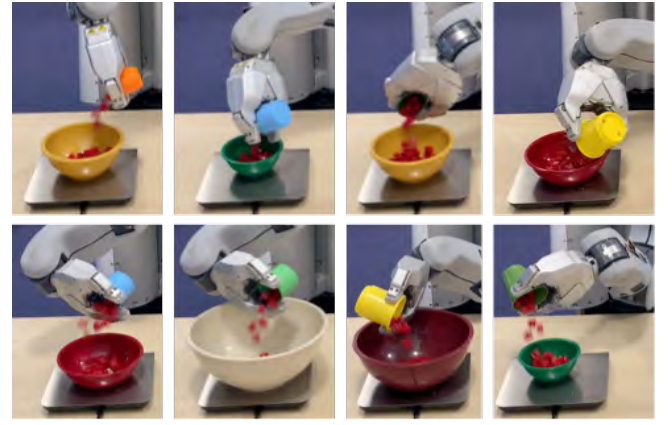
Figure 2. Examples of a real-world robot executing a trained pouring primitive in *KitchenPR2* for several contexts parameter values (cup dimensions) and control parameters values (relative cup poses).

3. Learning should be *sample efficient*: that is, it should require relatively few trial executions of the skill in different situations to acquire the models needed for planning. This is critical because of the high cost of running trials on a physical robot: not only must the robot execute the skill on each trial, it must set up the initial conditions appropriately for the next.

Figure 2 illustrates several instances of a parameterized sensorimotor policy for pouring with a real-world PR2. Context parameters for the skill encode the approximate dimensions of both the cup and bowl. Control parameters specify the cup's rotation about a coordinate frame, the final pitch of the cup in this frame, and the pose of this frame relative to the bowl. See figure 6 for a visualization of these parameters. Figure 3 displays the robot executing a sensorimotor policy for scooping. The objective of our work is to learn the set of sufficiently successful pours and scoops across a wide range of objects.

This paper is an extended version of Wang et al. (2018). Our new contributions include an in-depth discussion of how the learners, motion planners, and TAMP planner interact in our integrated system (section 2 and appendix B), the application of our approach to a high-dimensional robot manipulator operating in a 3D simulated dynamic manipulation environment (section 6), extensive experimentation within this environment that compares different learners, learning strategies, and sampling strategies (section 7), and finally real-world validation of the efficacy of learning individual primitives and deployment of the full system to solve multi-step manipulation problems (section 8).

In the rest of this paper, we 1) formulate a precise learning problem and explore algorithms based on Gaussian processes for efficiently learning models for and robustly applying robot skills (section 3); 2) formalize the overall problem of generating behavior involving these new skills using the PDDLStream TAMP planner (Garrett et al. 2020a) (section 4); and finally 3) present extensive empirical results both in physics simulations (sections 6 and 7) and on a real robot (section 8), demonstrating efficient model acquisition and robust use of new skills in complex problems.

Figure 3. Examples of a real-world robot executing a trained scooping primitive in *KitchenPR2* for several contexts (bowl and spoon dimensions) and control parameters (relative spoon poses).

2 Approach

We model the TAMP problem as one of controlling a robot operating in a deterministic discrete-time hybrid system. The state s of the system is comprised of a set of discrete and continuous state variables, which describe the properties and configuration of the robot as well as the objects in the environment. At each time step, the robot executes an action a, which corresponds to applying low-level motor torques. Let $\mathcal{S}$ be the state space of the system and $\mathcal{A}$ be the action space of the robot. The initial state of the system is $s_0 \in \mathcal{S}$, and the objective of the robot is to control the system to a state s_* contained with a specified set of goal states $s_* \in S_* \subseteq \mathcal{S}$. Let $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ be the system's transition function.

We assume that a set of parameterized skills $\mathcal{O}$ has already been programmed or learned, and it is our objective to learn a characterization of each skill that can be used by a task-and-motion planner. Each skill $O(\omega) \in \mathcal{O}$ is specified as an *option* (Sutton et al. 1999) with initiation set $I_{O(\omega)} \subseteq \mathcal{S}$, policy $\pi_{O(\omega)} : \mathcal{S} \to \mathcal{A}$, and termination set $G_{O(\omega)} \subseteq \mathcal{S}$ where Ω_O is the parameter space for skill O and $\omega \in \Omega_O$ is a particular parameter value. Let $\Gamma_O : \mathcal{S} \times \Omega_O \to \mathcal{S}$ be the option transition function for skill O where $\Gamma_O(s, \omega) = s'$ if and only if executing option $O(\omega)$ from state $s \in I_{O(\omega)}$ terminates in state $s' \in G_{O(\omega)}$, which is the result of recursively applying $s \leftarrow \mathcal{T}(s, \pi_{O(\omega)}(s))$ until reaching a termination state $s \in G_{O(\omega)}$. Controlling the system can now be viewed as a planning problem over skill instances where the objective is to find a finite sequence of k skill instances $O_1(\omega_1), ..., O_k(\omega_k)$ such that the corresponding sequence of states $s_0, s_1, ..., s_k$ satisfies $\Gamma_{O_i}(s_{i-1}, \omega_i) = s_i$ for $i \in \{1, ..., k\}$ as well as $s_k \in S_*$. Importantly, we consider the setting in which the robot *does not* have full knowledge of Γ_O, *i.e.* it does not have have a complete model of the effects of skill O. Without a model, the robot is unable to plan over sequences of different skill instances. Thus, we seek to learn

Γ_O for each skill O from data in order to combine and apply them during planning.

We learn Γ_O through estimating a constraint $\chi_O : \mathcal{S} \times \Omega_O \times \mathcal{S} \to \{0, 1\}$ where $\chi_O(s, \omega, s') = 1$ if and only if $\Gamma_O(s, \omega) = s'$. We will learn from training examples in the form of (s, ω, s') triplets, which represent a present state s, skill parameter value ω, and future state s'. In many systems, χ_O can be naturally expressed as a conjunction defined over a set of atomic constraints, each of which might only involve a small subset of the state variables and skill parameters present within (s, ω, s'). Additionally, we often have at least a partial model of Γ_O, namely we might know the analytic form of some constraints, such as geometric constraints involving kinematics and collisions, and thus need not redundantly learn these constraints from scratch. Finally, we will assume that the structure of each atomic constraint is given, meaning that we know which state variables and skill parameters are relevant for predicting the effects of the skill (but see work by Xia et al. (2019) for one strategy for learning the structure).

As an example, suppose the robot is given a skill whose intended effect is to pour the contents of a cup into a bowl, and we would like to learn the conditions under which executing that skill will transfer a sufficiently large fraction of the cup's initial contents into the target bowl. These conditions can be articulated as a constraint representing a relation among the initial poses of the cup and bowl, some aspects of their shapes, and the trajectory of the cup, defined in terms of waypoints and a final pose relative to the bowl. During planning, some of these state variable values, such as the dimensions of objects, are given by the problem and thus cannot be chosen by the robot. Thus, it is convenient to define constraint χ generically on (θ, α) parameter pairs instead of (s, ω, s') triplets where α are fixed *context* parameters and θ are *control* parameters that the robot can choose. These control parameters can include both parameters ω passed directly to the skill as well as additional aspects of the state s that the robot can control indirectly through the execution of other skills. It is important to note that the learned constraint does not apply directly to robot configurations: all operations will include default constraints on kinematic path existence and lack of collision that are based on the system's prior knowledge of robot motion. This means that the robot does not have to re-learn this foundational knowledge every time it acquires a new skill.

Once the set of constraints for each skill O are determined, a task-and-motion planner can construct a plan in the form of a sequence of these skill instances that achieves a desired goal condition. In order for a plan to be correct, a planner must ensure that the accompanying sequence of parameters and induced sequence of states satisfies the constraint χ_O for each skill O. Namely, a sequence of k skills $O_1, ..., O_k$ has an associated alternating sequence of states and skill parameters $s_0, \omega_1, s_1, ..., s_{k-1}, \omega_k, s_k$. Each contiguous triplet (s_{i-1}, ω_i, s_i) must satisfy $\chi_{O_i}(s_{i-1}, \omega_i, s_i) = 1$. A planner must search over sequences of skills as well as parameter values that satisfy these constraints. Finding values that satisfy these constraints is a nontrivial problem; however, existing work in TAMP has shown that a variety of methods can be effective (Garrett et al. 2021). In this work we take a sampling-based approach using the PDDLStream

TAMP planner (Garrett et al. 2020a), which we describe in appendix B.

3 Estimating the constraint

Our primary technical problem, then, is to learn a constraint representing the success criteria for executing a skill, represented as a relation among fixed context parameters α and free control parameters θ. For reasons outlined in the introduction, we seek to characterize the entire space of successful control parameters for any given context, using an explicit characterization of uncertainty in the learned relation to guarantee robust parameter selection at planning time.

3.1 Contextual super-level set estimation

We will focus on the formal problem of learning a function from values of the context parameters $\alpha \in \mathbb{R}^{d_\alpha}$ to sets of control parameters $\theta \in B$. We assume that the domain of θ is a hyper-rectangular space $B = [0, 1]^{d_\theta} \subset \mathbb{R}^{d_\theta}$, but generalization to other topologies is possible. We are interested in learning a Boolean function $\chi : B \times \mathbb{R}^{d_\alpha} \to \{0, 1\}$ for a skill of interest, where $\chi(\theta, \alpha) = 1$ if and only if executing the skill with control parameter θ and context parameter α results in the desired effect.

We assume that χ can be expressed in the form of an inequality constraint $\chi(\theta, \alpha) = [g(\theta, \alpha) > 0]$, where $g : B \times \mathbb{R}^{d_\alpha} \to \mathbb{R}$ is a real-valued scoring function with arguments θ and α. We denote the conditional *super-level set* of the scoring function given α by

$$A_\alpha \equiv \{\theta \in B \mid g(\theta, \alpha) > 0\}.$$

For example, the scoring function $g(\theta, \alpha)$ for pouring might be the proportion of poured liquid that actually ends up in the target cup, minus some target proportion. So, given any new situation with context parameters α, we know that any value of control parameters $\theta \in A_\alpha$ will result in success with high probability. This strategy relies on the availability of real-valued values of this score function during training rather than just binary labels of success or failure.

3.2 Active sampling for learning

Our objective in the learning phase is to efficiently gather data to characterize the conditional super-level sets A_α with high confidence. We use a Gaussian process (GP) on the score function g to select informative queries using a level-set estimation approach. In order to implement the constraint sampler, we must be able to sample from the super-level set A_α for any given context α During training, we select α values from a distribution reflecting naturally occurring contexts in the underlying domain, for example, the dimensions of cups and bowls in a pouring operation. In the event that the agent can initialize its environment, for example by picking the objects for an experiment, some "context parameters" can be viewed as control parameters that can be selected in the process of active learning. Note that learning an accurate description of the boundaries of the level-set is a different objective from learning all of the function values well and also different from finding the maximum function value, and so it must be handled differently from typical GP-based active learning.

For each α value in the training set, we apply the *straddle* algorithm (Bryan et al. 2006) to actively select samples of θ for evaluation by running the skill policy. After each new evaluation of $g(\theta, \alpha)$ is obtained, the data-set $\mathcal{D}$ is augmented with pair $\langle(\theta, \alpha), g(\theta, \alpha)\rangle$, and used to update the GP. Given the mean function $\mu(\cdot)$ and the variance function $\sigma^2(\cdot)$ for the posterior GP, the straddle algorithm selects θ that maximizes the *acquisition function*

$$\psi_{\mu,\sigma}(\theta, \alpha) = -|\mu(\theta, \alpha)| + 1.96\sigma(\theta, \alpha).$$

It has a high value for values of θ that are near the zero boundary for the given α or for which the score function is highly uncertain. The parameter 1.96 is selected such that if $\psi_{\mu,\sigma}(\theta, \alpha)$ is negative, θ has less than 5 percent chance of being in the level set. In practice, this heuristic has been observed to deliver state-of-the-art learning performance for level set estimation (Bogunovic et al. 2016; Gotovos et al. 2013). After each new evaluation, we retrain the Gaussian process by maximizing its marginal data-likelihood with respect to its hyper-parameters. Algorithm 1 specifies the algorithm; GP-PREDICT($\mathcal{D}$) computes the posterior mean and variance, which is explained in appendix A.

Algorithm 1 Active Bayesian Level Set Estimation

1: Given initial data set $\mathcal{D}$, context α, number of samples T
2: **for** $t \in \{1, ..., T\}$ **do**
3:   $\mu, \sigma \leftarrow$ GP-PREDICT($\mathcal{D}$)
4:   $\theta \leftarrow \arg\max_\theta \psi_{\mu,\sigma}(\theta, \alpha)$
5:   $y \leftarrow g(\theta, \alpha)$
6:   $\mathcal{D} \leftarrow \mathcal{D} \cup \{\langle(\theta, \alpha), y\rangle\}$
7: **return** $\mathcal{D}$

---

4 Planning with a new skill

We have shown how to take a controller for a new motor skill and use active-learning strategies to estimate a constraint representing the conditions under which executing that skill will have a desired effect. In this section, we describe our strategies for integrating that new skill into a TAMP system.

4.1 The need for sampling during planning

Planning for TAMP problems is difficult, because it requires integrating aspects of motion planning through continuous robot configuration space, AI-style planning through discrete choices of operations and objects, and the selection of real-valued parameters, such as object grasps and placements as well as robot configurations that enable the execution of manipulation operations.

In our work, we use the PDDLStream planning framework (Garrett et al. 2020a), which is discussed in more detail in appendix B. In this framework, a skill description must specify the constraint on parameter values and a *sampler* that can, given values of context parameters α, produce a stream of assignments to the control parameters θ. In this section, we focus on the construction and use of this sampler.

The reason for sampling values of θ, rather than simply selecting the one that maximizes the likelihood of success given α, is that there may be other considerations that make θ infeasible in broader planning context. For example,

a particular grasp of the cup to be poured from might acceptable for pouring, but unreachable for the robot given the current placement of the object on the table.

Our objective in the planning phase is to select a diverse set of samples $\{\theta_i\}$ for which it is likely that (α, θ_i) satisfy both the learned constraint χ and the rest of the constraints in the planner. We do this in two steps: first, we use a novel risk-aware sampler to generate θ values that satisfy the learned constraint with high probability; second, we integrate this sampler with PDDLStream, where we generate samples from this set that represent its diversity, in order to expose the full variety of choices to the planner.

4.2 Risk-aware sampling

We can use our Bayesian estimate of the scoring function g to select action instances for planning. Given a new context α, which need not have occurred in the training set—the GP will provide generalization over contexts—we would like to sample a sequence of $\theta \in B$ such that with high probability, $g(\theta, \alpha) > 0$. In order to guarantee this, we adopt a concentration bound and a union bound on the predictive scores of the samples. Notice that by construction of the GP, the predictive scores are Gaussian random variables. Letting $\phi_{\mu,\sigma}(\theta, \alpha)$ be the ratio of the predicted mean and standard deviation,

$$\phi_{\mu,\sigma}(\theta, \alpha) = \mu(\theta, \alpha)/\sigma(\theta, \alpha),$$

the following is a direct corollary of lemma 3.2 of Wang et al. (2016):

Corollary 1. *Let $g(\theta, \alpha) \sim \mathrm{GP}(\mu, \sigma)$, and for $\delta \in (0, 1)$ set $\beta_i^* = \sqrt{2 \log(\pi_i/2\delta))}$, where $\sum_{i=1}^{T} \pi_i^{-1} \le 1$, $\pi_i > 0$.*
If $\forall i \in \{1, ..., T\}$ $\phi_{\mu,\sigma}(\theta_i, \alpha) > \beta_i^*$,
then $\Pr[g(\theta_i, \alpha) > 0, \forall i] \ge 1 - \delta$.

Corollary 1 enables us to properly construct the set of parameters that satisfy the inequality constraint $g(\theta, \alpha) > 0$ with high probability. Here we define the *high-probability super-level set* of θ for context α as

$$\hat{A}_\alpha = \{\theta \mid \phi_{\mu,\sigma}(\theta, \alpha) > \beta^*\}$$

where β^* is picked according to corollary 1. If we draw T samples from $\hat{A}_\alpha$, then with probability at least $1 - \delta$, all of the samples will satisfy the constraint $g(\theta, \alpha) > 0$.

In practice, however, for a given α and using the definition of β^* from corollary 1, the set $\hat{A}_\alpha$ may be empty. To account for this, we relax our criterion to include the set of θ values whose score is within 5% of the value of the most confident parameter, and define an alternative score threshold $\beta = \Phi^{-1}(0.95\Phi(\phi_{\mu,\sigma}(\theta^*, \alpha))$ where Φ is the cumulative density function of a normal distribution and θ^* is the *most confident parameter*, *i.e.*

$$\theta^* = \arg\max_{\theta \in B} \phi_{\mu,\sigma}(\theta, \alpha).$$

Although we can obtain the derivatives of function $\phi_{\mu,\sigma}(\cdot)$, we may not be able to solve the optimization problem due to the multi-modality of this function. However, we can approximate the solution to the global optimization of function $\phi_{\mu,\sigma}(\cdot)$ over domain B by restarting gradient-based optimization at a few locations within domain B.
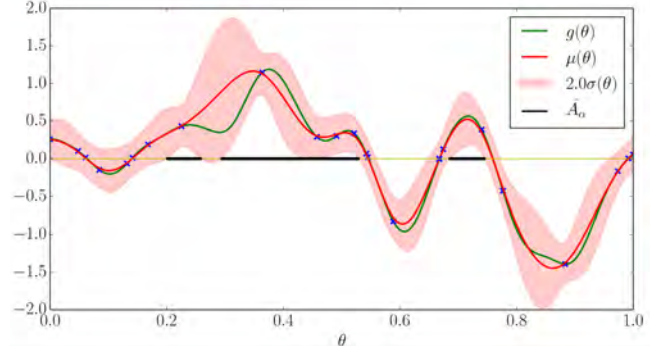
Figure 4. High-probability super-level set in black.

Alternatively, we may estimate θ^* by sampling a set of n parameters $\{\theta_1, ..., \theta_n\} \in B$, and returning the value $\theta^* = \arg\max_{\theta_i} \phi_{\mu,\sigma}(\theta_i, \alpha)$.

Figure 4 illustrates the computation of $\hat{A}_\alpha$. The green line is the true hidden $g(\theta)$; the blue $\times$ symbols are the training data, gathered using the straddle algorithm in $[0, 1]$; the red line is the posterior mean function $\mu(\theta)$; the pink regions show the two-standard-deviation bounds on $g(\theta)$ based on $\sigma(\theta)$; and the black line segments are the high-probability super-level set $\hat{A}_\alpha$ for $\beta = 2.0$. We can see that sampling has concentrated near the boundary, that $\hat{A}_\alpha$ is a subset of the true super-level set, and that as σ decreases through experience, $\hat{A}_\alpha$ will approach the true super-level set.

4.3 Efficient adaptive sampling

To sample from $\hat{A}_\alpha$, one simple strategy is to do rejection sampling with a proposal distribution that is uniform on the search bounding-box B. However, in many cases, the feasible region of a constraint is much smaller than B, which means that uniform sampling will have a very low chance of drawing samples within $\hat{A}_\alpha$, and so rejection sampling will be very inefficient. We address this problem using a novel adaptive sampler, which draws new samples from the neighborhood of the samples that are already known to be feasible with high probability and then re-weights these new samples using importance weights.

The algorithm ADAPTIVESAMPLER in algorithm 2 takes as input the posterior GP parameters μ and σ and context vector α, and yields a stream of samples. It begins by computing β, then sets Θ_{init} to contain the θ that is most likely to satisfy the constraint. It then maintains a buffer Θ of at least $m/2$ samples and yields the first one each time it is required to do so; it technically never actually returns, but yields a sample each time it is queried.

Algorithm 2 Super-level Set Adaptive Sampling

1: **function** ADAPTIVESAMPLER(μ, σ, α)
2:   $\Theta \leftarrow \emptyset$
3:   $\beta \leftarrow \Phi^{-1}(0.95\Phi(\max_{\theta \in B} \phi_{\mu,\sigma}(\theta, \alpha)))$
4:   $\Theta_{init} \leftarrow \{\arg\max_{\theta \in B} \phi_{\mu,\sigma}(\theta, \alpha)\}$
5:   **while True do**
6:     **if** $|\Theta| < m/2$ **then**
7:       $\Theta \leftarrow$ SAMPLEBUFFER($\mu, \sigma, \alpha, \beta, \Theta_{init}, n, m$)
8:     $\theta \leftarrow \Theta[0]$
9:     **yield** θ
10:    $\Theta \leftarrow \Theta \setminus \{\theta\}$

The main work is done by SAMPLEBUFFER in algorithm 3, which constructs a mixture of truncated Gaussian distributions (TGMM), specified by mixture weights p, means Θ, circular variance with parameter v, and bounds B. Parameter v indicates how far from known good θ values it is reasonable to search; it is increased if a large portion of the samples from the TGMM are accepted and decreased otherwise. The algorithm iterates until it has constructed a set of at least m samples from $\hat{A}_\alpha$. It samples n elements from the TGMM and retains those that are in $\hat{A}_\alpha$ as Θ_a. Then, it computes "importance weights" p_a that are inversely related to the probability of drawing each $\theta_a \in \Theta_a$ from the current TGMM. This will tend to spread the mass of the sampling distribution away from the current samples, but still keep it concentrated in the target region. A set of n uniform samples is drawn and filtered, again to maintain the chance of dispersing to good regions that are far from the initialization. The p values associated with the old Θ as well as the newly sampled ones are concatenated and then normalized into a distribution, the new samples added to Θ, and the loop continues. When at least m samples have been obtained, m elements are sampled from Θ according to distribution p, without replacement.

Algorithm 3 Sampling From a Truncated Gaussian Buffer

1: **function** SAMPLEBUFFER($\mu, \sigma, \alpha, \beta, \Theta_{init}$)
2: $\quad \Theta \leftarrow \Theta_{init}$
3: $\quad v \leftarrow [1]_{d=1}^{d_\theta}; p \leftarrow [1]_{i=1}^{|\Theta|}$
4: $\quad$ **while** True **do**
5: $\quad\quad \Theta' \leftarrow$ SAMPLETGMM($n; p, \Theta, v, B$)
6: $\quad\quad \Theta_a \leftarrow \{\theta \in \Theta' \mid \phi_{\mu,\sigma}(\theta, \alpha) > \beta\}$
7: $\quad\quad p_a \leftarrow 1/p_{\text{TGMM}}(\Theta_a; p, \Theta, v, B)$
8: $\quad\quad v \leftarrow v/2$ **if** $|\Theta_a| < |\Theta'|/2$ **else** $2v$
9: $\quad\quad \Theta'' \leftarrow$ SAMPLEUNIFORM($n; B$)
10: $\quad\quad \Theta_r \leftarrow \{\theta \in \Theta'' \mid \phi_{\mu,\sigma}(\theta, \alpha) > \beta\}$
11: $\quad\quad p_r \leftarrow [Vol(B)]_{i=1}^{|\Theta_r|}$
12: $\quad\quad p \leftarrow$ NORMALIZE($[p, p_r, p_a]$)
13: $\quad\quad \Theta \leftarrow [\Theta, \Theta_r, \Theta_a]$
14: $\quad\quad$ **if** $|\Theta| > m$ **then**
15: $\quad\quad\quad$ **return** SAMPLE($m; \Theta, p$)

It is easy to see that as n goes to infinity, by sampling from the discrete set according to the re-weighted probability, we are essentially sampling uniformly at random from $\hat{A}_\alpha$. This is because $\forall \theta \in \Theta$, $p(\theta) \propto \frac{1}{p_{sample}(\theta)} p_{sample}(\theta) = 1$. For uniform sampling, $p_{sample}(\theta) = \frac{1}{Vol(B)}$, where $Vol(B)$ is the volume of B; and for sampling from the truncated mixture of Gaussians, $p_{sample}(\theta)$ is the probability density of θ. In practice, of course, n is finite, but this method is much more efficient than rejection sampling.

4.4 Diversity-aware sampling for planning

Now that we have a sampler that can generate approximately uniformly random samples within the region of values that satisfy the constraints with high probability, we can use it inside a planning algorithm to explore continuous action spaces. A planner may need to consider multiple different parameterized instances of a particular action before finding one that satisfies all the constraints in the overall context of the planning problem. For example, some good pours may not be kinematically reachable given the robot's current configuration, so sampling a single pour might be insufficient for solving the task.

The efficiency of this planning process depends on the order in which samples are generated. Intuitively, when previous samples for a context parameter have failed to contribute to a successful plan, it would be wise to try new samples that, while still having high probability of satisfying the constraint, are as different as possible from those that were previously tried. We need, therefore, to consider diversity when generating samples; but the precise characterization of useful diversity depends on the domain in which the method is operating. We address this problem by adapting a kernel that is used in the sampling process, based on experience in previous planning problems.

Diversity-aware sampling has been studied extensively with determinantal point processes (DPPs) (Kulesza et al. 2012). We begin with similar ideas and adapt them to our planning domain, quantifying the diversity of a set of samples S using the determinant of a Gram matrix

$$D(S) = \log\det(\Xi^S \zeta^{-2} + \boldsymbol{I}),$$

where $\Xi_{ij}^S = \xi(\theta_i, \theta_j)$ for $\theta_i, \theta_j \in S$, ξ is a covariance function, and ζ is a free parameter (we use $\zeta = 0.1$). In DPPs, the quantity $D(S)$ can be interpreted as the volume spanned by the feature space of the kernel $\xi(\theta_i, \theta_j)\zeta^{-2} + \boldsymbol{1}_{\theta_i \equiv \theta_j}$ assuming that $\theta_i = \theta_j \iff i = j$. Alternatively, one can interpret the quantity $D(S)$ as the information gain of a GP when the function values on S are observed (Srinivas et al. 2010). This GP has kernel ξ and observation noise $\mathcal{N}(0, \zeta^2)$. Because of the submodularity and monotonicity of $D(\cdot)$, we can maximize $D(S)$ greedily with the promise that

$$D([\theta_i]_{i=1}^N) \geq (1 - \frac{1}{e}) \max_{|S| \leq N} D(S)$$

$\forall N = 1, 2, \ldots$ where $\theta_i = \arg\max_\theta D(\theta \cup \{\theta_j\}_{j=1}^{i-1})$. In fact, maximizing $D(\theta \cup S)$ is equivalent to maximizing

$$\eta_S(\theta) = \xi(\theta, \theta) - \boldsymbol{\xi}^S(\theta)^{\mathrm{T}}(\Xi^S + \zeta^2 \boldsymbol{I})^{-1}\boldsymbol{\xi}^S(\theta)$$

which is exactly the same as the posterior variance for a GP.

The DIVERSESAMPLER procedure is very similar in structure to the ADAPTIVESAMPLER procedure, but rather than selecting an arbitrary element of Θ, the buffer of good samples, we track the set S of samples that have already been returned and select the element of Θ that is most diverse from S as the sample to yield on each iteration. In addition, we yield S to enable kernel learning as described in Algorithm 5, to yield a kernel η.

Algorithm 4 Super-level Set Diverse Sampling

1: **function** DIVERSESAMPLER($\mu, \sigma, \alpha, \eta$)
2: $\quad \Theta \leftarrow \emptyset; S \leftarrow \emptyset$
3: $\quad \theta \leftarrow \arg\max_{\theta \in B} \phi_{\mu,\sigma}(\theta, \alpha)$
4: $\quad \beta \leftarrow \lambda(\phi_{\mu,\sigma}(\theta, \alpha))$
5: $\quad$ **while** planner requires samples **do**
6: $\quad\quad$ **yield** θ, S
7: $\quad\quad$ **if** $|\Theta| < m/2$ **then**
8: $\quad\quad\quad \Theta \leftarrow$ SAMPLEBUFFER($\mu, \sigma, \alpha, \beta, \Theta_{init}$)
9: $\quad\quad S \leftarrow S \cup \{\theta\}$ $\qquad \triangleright$ S contains samples before θ
10: $\quad\quad \theta \leftarrow \arg\max_{\theta \in \Theta} \eta_S(\theta)$
11: $\quad\quad \Theta \leftarrow \Theta \setminus \{\theta\}$

It is typical to learn the kernel parameters of a GP or DPP given supervised training examples of function values or diverse sets, but those are not available in our setting; we can only observe whether the set of samples is sufficient for the planner to identify a solution. We derive our notion of similarity by assuming that all samples that fail to lead to a solution are similar. Under this assumption, we develop an online learning approach that adapts the kernel parameters to learn a good diversity metric for a sequence of planning tasks. We use the FOCUSED algorithm (appendix B.3) as our PDDLStream planner in order to more precisely determine which sampled values failed to satisfy a downstream plan constraint for a particular plan skeleton.

We use the squared exponential kernel of the form $\xi(\theta, \gamma; l) = \exp(-\sum_d r_d^2)$, where $r_d = |l_d(\theta_d - \gamma_d)|$ is the rescaled "distance" between θ and γ on the d-th feature and l is the inverse length scale. Let θ be the sample that failed and the set of samples sampled before θ be S. We define the importance of the d-th feature as

$$\tau_S^\theta(d) = \xi(\theta_d, \theta_d; l_d) - \boldsymbol{\xi}^S(\theta_d; l_d)^{\mathrm{T}} (\Xi^S + \zeta^2 \boldsymbol{I})^{-1} \boldsymbol{\xi}^S(\theta_d; l_d),$$

which is the conditional variance if we ignore the distance contribution of all other features except the d-th; that is, $\forall k \neq d, l_k = 0$. Note that we keep $\Xi_i + \zeta^2 \boldsymbol{I}$ the same for all the features so that the inverse only needs to be computed once.

The diverse sampling procedure is analogous to the weighted majority algorithm (Foster and Vohra 1999) in that each feature d is seen as an expert that contributes to the conditional variance term, which measures how diverse θ is with respect to S. The contribution of feature d is measured by $\tau_S^\theta(d)$. If θ was rejected by the planner, we decrease the inverse length scale l_d of feature $d = \arg\max_{d \in [d_\theta]} \tau_S^\theta(d)$ to be $(1 - \epsilon)l_d$, because feature d contributed the most to the decision that θ was most different from S.

Algorithm 5 Task-level Kernel Learning

1: **for** task in T **do**
2:      $S \leftarrow \emptyset$
3:      $\alpha \leftarrow$ current context
4:      $\mu, \sigma \leftarrow$ GP-PREDICT(α)
5:      **while** plan not found **do**
6:          **if** $|S| > 0$ **then**
7:              $d \leftarrow \arg\max_{d \in [d_\theta]} \tau_S^\theta(d)$
8:              $l_d \leftarrow (1 - \epsilon)l_d$
9:          $\theta, S \leftarrow$ DIVERSESAMPLER($\mu, \sigma, \alpha, \xi(\cdot, \cdot; l)$)
10:          Check if a plan exist using θ

Algorithm 5 depicts a scenario in which the kernel is updated during interactions with a planner; it is simplified in that it uses a single sampler, but in our experimental applications there are many instances of action samplers in play during a single execution of the planner. Given a sequence of tasks presented to the planner, we can continue to apply this kernel update, molding our diversity measure to the demands of the distribution of tasks in the domain. This simple strategy for kernel learning may lead to a significant reduction in planning time, as we demonstrate in Section 7.

5 Related work

Our work draws ideas from model learning, probabilistic modeling of functions, and task and motion planning.

There is a large amount of work on learning individual motor primitives such as pushing (Kroemer and Sukhatme 2016a; Hermans et al. 2013), scooping (Schenck et al. 2017), and pouring (Pan et al. 2016; Tamosiunaite et al. 2011; Brandi et al. 2014; Yamaguchi and Atkeson 2016; Schenck and Fox 2017). We focus on the task of learning models of these primitives suitable for multi-step planning. We extend a particular formulation of planning-model learning (Kaelbling and Lozano-Perez 2017), where constraint-based preimage models are learned for parameterized action primitives, by giving a probabilistic characterization of the preimage and using these models during planning.

There are several other approaches for learning precondition and effect models of sensorimotor skills that are suitable for planning. Konidaris et al. (2018) construct a completely symbolic model of skills that enables purely symbolic task planning. Our method, on the other hand, learns hybrid models, involving continuous parameters. Kroemer and Sukhatme (2016b) learn image classifiers for preconditions but do not support general-purpose planning. More recently, Wang and Kroemer (2019) learn state transition models for sequencing low-level motor skills that perform manipulation tasks.

We use GP-based level-set estimation (Bryan et al. 2006; Gotovos et al. 2013; Rasmussen and Williams 2006; Bogunovic et al. 2016) to model the feasible regions (superlevel set of the scoring function) of action parameters. We use the *straddle* algorithm (Bryan et al. 2006) to actively sample from the function threshold, in order to estimate the superlevel set that satisfy the constraint with high probability. Our methods can be extended to other function approximators that give uncertainty estimates, such as Bayesian neural networks and their variants (Gal and Ghahramani 2016; Lakshminarayanan et al. 2017).

Alternatively, one can use GP classification methods with active learning (Kapoor et al. 2007) to model our constraints. Active learning of GP classifiers is often used for modeling safety constraints to help perform safe exploration (Schreiter et al. 2015; Englert and Toussaint 2016). The focus of this work, however, is to present a suite of approaches to address not only how to actively learn a model but also how to use learned models to solve complex long-horizon manipulation tasks. In this work, we only focus on one setting of the active model learning problem (level set estimation with GP regression) but other active learning approaches can certainly be used.

Determinantal point processes (DPPs) (Kulesza et al. 2012) are typically used for diversity-aware sampling. However, both sampling from a continuous DPP (Hafiz Affandi et al. 2013) and learning the kernel of a DPP (Affandi et al. 2014) are challenging.

Several approaches to TAMP utilize generators to enumerate infinite sequences of values (Kaelbling and Lozano-Pérez 2011; Srivastava et al. 2014; Garrett et al. 2017a). Our learned samplers can be incorporated into any of these approaches. Additionally, some recent papers have

investigated learning effective samplers within the context of TAMP. Chitnis et al. (2016) frame learning plan parameters as a reinforcement-learning problem and learn a randomized policy that samples from a discrete set of robot base and object poses. Kim et al. (2017) proposed a method for selecting from a discrete set of samples by ranking new samples based on their correlation with previously attempted samples. In subsequent work, they instead train a generative adversarial network (GAN) to directly generate a distribution of satisfactory samples (Kim et al. 2018).

6 Experimental domains

We analyze the effectiveness and efficiency of each component of our system independently and then demonstrate their collective performance in the context of planning for long-horizon tasks in a simulated high-dimensional manipulation domain. We have carried out experiments in three settings:

- *Kitchen2D*: a simulated 2D kitchen domain implemented in Box2D (Catto 2011); a description of the simulation and results can be found in appendix C and in our earlier paper (Wang et al. 2018).
- *Kitchen3D*: a new simulated 3D kitchen domain implemented in PyBullet (Coumans and Bai 2016–2019); a description of the simulation and results are given in this section.
- *KitchenPR2*: a real-world experiment with a PR2 robot; a description of the implementation and results are given in section 8.

6.1 Implementation of Kitchen3D

To investigate how well our approach scales to high-dimensional robots interacting with 3D objects, we implemented a simulated 3D tabletop environment with a dual-arm PR2 robot. The 3D environment serves to bridge the gap between our previous 2D domain and a real-world robot operating scenario. Our 3D simulation uses the PyBullet (Coumans and Bai 2016–2019) physics engine. An illustration of the robot performing pouring and scooping skills is shown in figure 5.

We experimented using objects created by randomly adapting meshes from our real-world data set of bowls, cups, and spoons, illustrated in figure 16. We uniformly-at-random and independently scale the diameter and height of each bowl and cup, but do not geometrically alter the three spoons. We randomly sample mass, inertial, damping, and frictional properties for all objects according to a truncated Gaussian distribution. Finally, we randomly sample the number, radius, and density of the spherical liquid particles. This randomization process ensures that with probability one, each training or testing trial is unique.

We use PyBullet not only during simulation but also during planning for forward kinematics, collision checking, and visualization. We plan for each of the PR2's two 7 degree of freedom robot arms independently. We use IKFast (Diankov and Kuffner 2008; Diankov 2010) for inverse kinematics. We use RRT-Connect (Kuffner Jr. and LaValle 2000) to plan free-space arm motions. Finally, we use Randomized Gradient Descent (RBD) (Yao and Gupta 2005; Stilman 2010), a constrained motion planner for

planning robot joint motions that follow a Cartesian gripper path.

In order to transport and dump particles from a cup or spoon into a bowl, the robot must ensure that the cup or spoon does not spill any of the particles during transit. To enforce this, we impose constraints $|\rho(q)| \leq \pi/6, |\phi(q)| \leq \pi/6$ that the grasped object's orientation remain within a safe region whenever the robot is carrying an object that contains particles, where $\rho(q), \phi(q)$ give the roll and pitch of the tool at configuration q. This constraint can be easily incorporated into robot motion planning by adding an additional check within the configuration "collision" function.

The robot executes actions by following planned sequences of arm or gripper configurations using a position controller. We apply a rigid attachment constraint whenever the robot intentionally grasps an object to better model the real world, where the robot can reliably move without the held object deviating significantly relative to its hand.

We focus on learning conditional samplers for pouring and scooping because they are the most challenging to learn due to particle dynamics. Similar to our work in *Kitchen2D*, we score pours and scoops relative to the filled capacity of the involved bowl or spoon. We approximately compute the total number of particles that successfully ended up in a bowl, cup, or spoon by counting the number of particles contained within the 3D axis-aligned bounding box of these objects at the end of simulation. We use a piecewise linear scoring function with threshold hyperparameter $\tau \in (0, 1)$ defined on the fraction of particles filled $x \in [0, 1]$:

$$g(x; \tau) = \begin{cases} -1 + x/\tau & 0 \leq x \leq \tau \\ (x - \tau)/(1 - \tau) & \tau < x \leq 1 \end{cases}.$$

This function chosen for the following properties: $g(0; \tau) = -1$, $g(\tau; \tau) = 0$, and $g(1; \tau) = +1$. For pouring, x is the final number of particles in the bowl over the initial number of particles in the cup, and we used $\tau = 0.9$. For scooping, x is the final number of particles in the spoon relative to the capacity of the spoon, and we used $\tau = 0.7$.

We assume that bowls and cups are approximately cylindrically symmetric, allowing us to parameterize contexts and controls using radial (r) and z coordinates. During learning, we use min-max normalization to scale each parameter value to within the interval $[-1, +1]$.
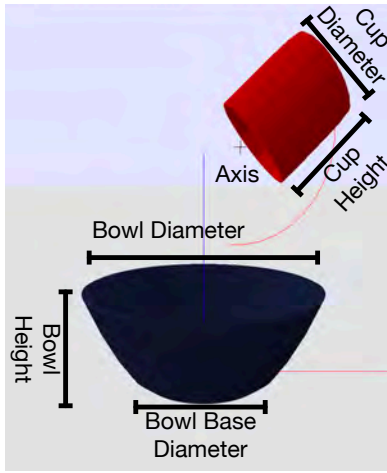
6.2 Parameterization

For both pouring and scooping, we derive context parameters from the base diameter, top diameter, and height of a bowl or cup. For scooping, we consider an additional context parameter for the length of a spoon. As a result, pouring has 6 context parameters, and scooping has 4 context parameters.

Our control parameterization determines a sequence of waypoints that the cup or spoon moves through. Then, we interpolate though these waypoints to obtain the full path of the moving object. We parameterized controls to be relative to the center of the base of a bowl or cup. The pouring control parameters are the initial upright r, z position of the cup relative to the bowl, the r, z point for the cup to rotate about relative to its initial position, and the final pitch of the cup. The scooping control parameters are the initial downward-facing r, z position of the spoon relative

Figure 5. Scenes of a simulated PR2 robot solving planning tasks requiring pouring as well as stacking (*left*), pushing (*center*), and making coffee (*right*).

Figure 6. A visualization of the context parameters (the bowl and cup dimensions) and control parameters (the axis of rotation, the cup rotation frame, and the final pitch), for a pour. The red curve is the path of the cup base during the pour.

to the bowl, the r scoop distance, and the final r, z upright position of the spoon. Thus, pouring and scooping both have 5 control parameters. We normalize distance-related control parameters relative to a bowl context parameter defined on the same coordinate in order to make the parameter space invariant to size of the involved bowl. This ensures that uniform exploration of the prediction space produces roughly the same frequency of successful pours across different bowl sizes. Figure 6 visualizes the context and control parameters for a pour. Figure 7 demonstrates the robot executing sampled pouring and scooping actions in simulation.

Figure 7. The simulated robot executing simulated pour (*left*) and scoop (*right*) actions.

We specify additional constraints per skill that enforce that execution does not knowingly cause any undesirable consequences. In our application, we prohibit any unsafe contact between objects; however, this function can be any general-purpose test. For pouring, this constraint enforces that full cup trajectory must not collide with the bowl. For scooping, this constraint enforces that the final spoon pose must not collide with the bowl. Satisfying the hard constraint function does not guarantee that the planner will able to find a full collision-free robot path to execute the path specified by the control parameters. For example, a proposed pour in the interior of a bowl might not collide with the bowl; however, it might not admit any collision-free grasps of the cup. Because this failure can be evaluated during planning, the control parameter during training should not receive the same negative score as a pour whose low quality can only be deduced after execution. Thus, we weakly penalize learner predictions for which we failed to find plans with a small negative score, reflecting the computational time wasted by considering that sample.

## 7 Experiments in simulation

We evaluated the performance of our approach in the *Kitchen3D* environment. See appendix C.2 for several additional experiments performed in our *Kitchen2D* environment (Wang et al. 2018).

7.1 Supervised learning

To aid with training and evaluating models in *Kitchen3D*, we first collected 10,000 pour and scoop trials by sampling a context and control parameter uniformly at random. These examples are used for training traditional (non-active) learners, for holdout test evaluation, and for efficiently approximating active learning (as described in section 7.2).

We first compared the performance of a GP using the multi-layer perceptron kernel trained *without* active learning with four baselines available through SKLearn (Pedregosa et al. 2011): (1) a neural network classifier (NNc), (2) a neural network regressor (NNr), (3) a random forest classifier (RFc), (4) and a random forest regressor (RFr). The following section describes two experiments per skill, which compare the likelihood of success and classification coverage across the decision space.

7.1.1 Success rate: First, we compared the average success rate of the single best prediction per learner. We
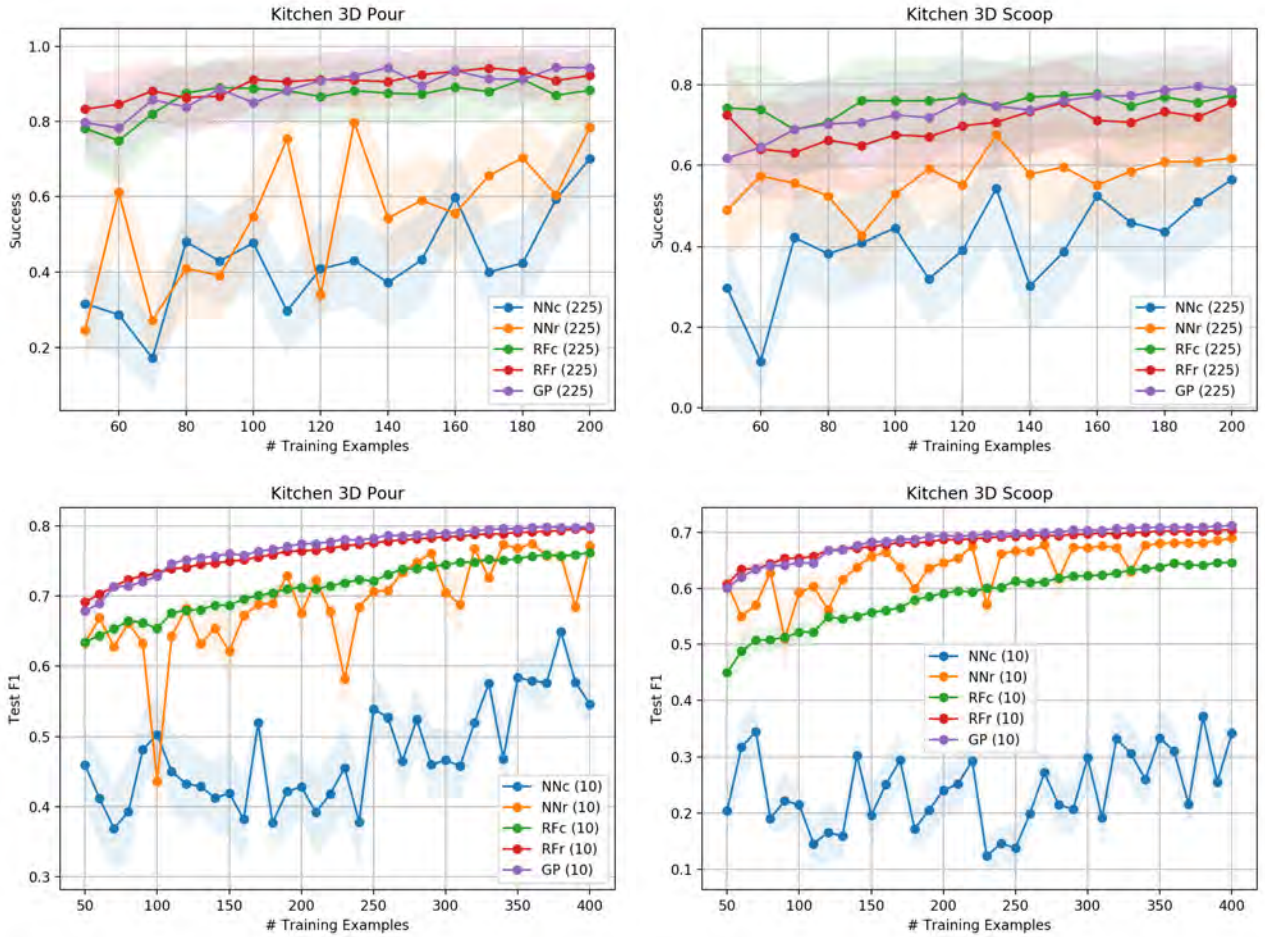
Figure 8. Pouring and scooping learning curves comparing a neural network classifier (NNc), a neural network regressor (NNr), a random forest classifier (RFc), a random forest regressor (RFr), and a GP using the multi-layer perceptron kernel. *Top row*: the success rate of the most confident control parameter produced by each learner. *Bottom row*: the test F1 score for each learner.

trained each learner on 5 randomly shuffled sequences of 200 training examples. We evaluated the success rate after every 10 examples by sampling 45 contexts, optimizing for the best control parameter per context, simulating the control parameter, and scoring the outcome. For the SKLearn classifiers, the best control parameter was obtained by maximizing the probability that the parameter is successful. For the SKLearn regressors, the best control parameter was obtained by maximizing the predicted score for the parameter. To optimize these scores, we randomly sampled 1,000 control parameters, sorted them in order of decreasing score, and returned the first control parameter that respects the hard constraints (described in section 6.1). Finally, we treated the high-probability parameter that maximizes equation 4.2 as the best control parameter, which incorporates both the predicted mean and standard deviation.

Figure 8 (*top*) shows the success rate learning curve*. The random forest and GP methods greatly outperform the neural network methods. Additionally, the GP ultimately achieves the best average success rate, likely due to its awareness of its own uncertainty.

7.1.2 F1 score: Second, we compared the F1[†] classification score on held-out test data. We trained each model on 10 randomly shuffled train and test splits, each consisting of 400 training examples and 1000 test examples. The predicted

label for a classifier is simply the most likely class, and the predicted label for a regressor is positive if the expected score is positive. Figure 8 (*bottom*) displays the test F1 score learning curves. The regressors outperform the classifiers, despite the fact the models were evaluated using the F1 score, a classification metric. This is likely because the underlying score functions are real-valued. When near the zero level set, small changes in score, which may be due to simulation noise caused by latent physical properties, can change the binary label of the example. As a result, these regions may have high variance due to the strict thresholding. Thresholding the score would be particularly detrimental when estimating uncertainty using a GP, as these regions have substantial uncertainty that is not derived from the underlying stochastic process but rather from the nature of thresholding. An active learner trained on thresholded score might indefinitely select examples near the zero level set because the process noise there is much larger than the rest of the space.

7.1.3 Kernel selection: Finally, we compared the GP performance when trained on the three kernels described in appendix A: the squared exponential radial basis kernel (GP-RBF), the Matérn kernel (GP-Matern52), and the

---

*This and all other *Kitchen3D* plots have 1/4 standard error shaded.
[†]The F1 score is the harmonic mean of the precision and recall of a test.
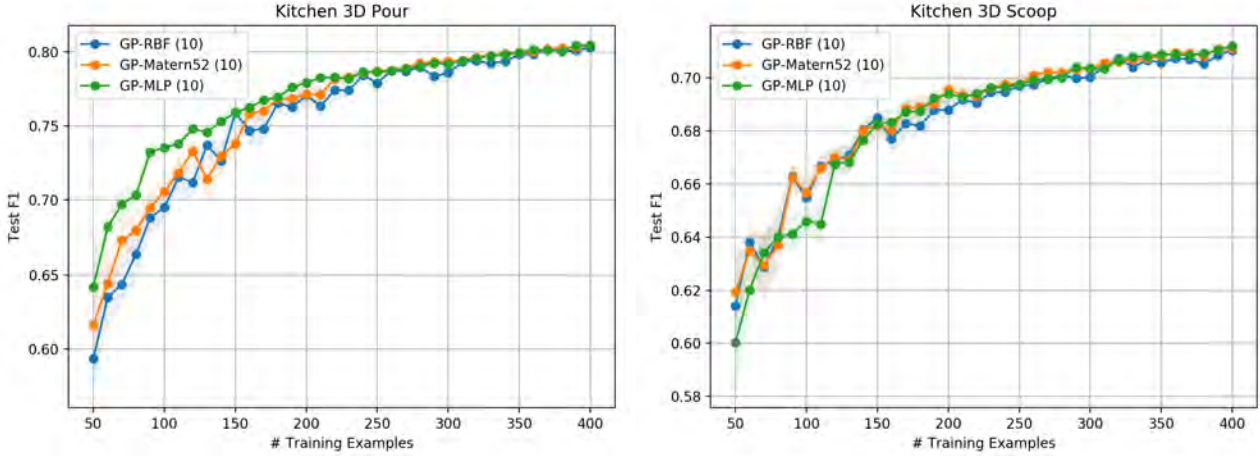
Figure 9. The test F1 score of the GP when trained with the squared exponential, Matérn, and multi-layer perceptron kernels (section A).

less commonly used multi-layer perceptron kernel (GP-MLP). Figure 9, compares the F1 test performance when experimenting with each of kernels, using the same conditions as described in section 7.1.2. The multi-layer perceptron kernel slightly outperforms both the squared exponential and Matérn kernels. We hypothesize that is due to the discontinuous nature of the pouring and scooping scoring functions; the score of a pour or scoop can vary dramatically when, for instance, the pour ejects particles near an edge of the bowl.
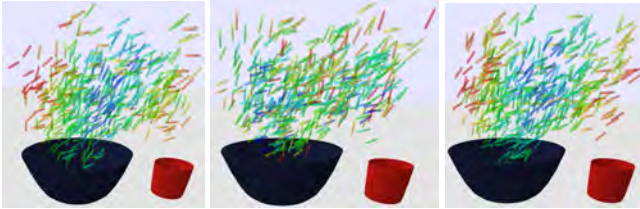
Figure 10. A visualization of the final cup pose for 500 pour valid control parameters. Poses are colored according to normalized mean (μ), inverse standard deviation ($1/\sigma$), and best probability (μ/σ) GP predictions, where red poses have the smallest values and blue poses have the largest values.

7.1.4 Visualizing predictions: We created a geometric visualization for the trained GP's mean and standard deviation score predictions across the space of legal control parameters. Figure 10 renders a data set of pour control parameters for a single bowl and cup pair by displaying the final pose of the red cup. It visualizes the GP's mean, inverse standard deviation, and most confident predictions by coloring small values red and large values blue. The mean predictions (*left*) demonstrate that the model learns that the z-axis of the cup must roughly intersect with the interior of the bowl for a pour to be successful. The standard deviation predictions (*center*) suggest that the more negative the cup pitch is, the higher variance in the outcome. We hypothesize that this is because the longer rotation ejects the particles at larger velocities, making particles more likely to bounce out of the bowl. The most confident prediction (*right*) combines the mean and standard deviation predictions. Incorporating the standard deviation biases the learner towards high scoring

pours that are closer to level. These results suggest that the GP is capturing intuitively relevant information for a successful pour.
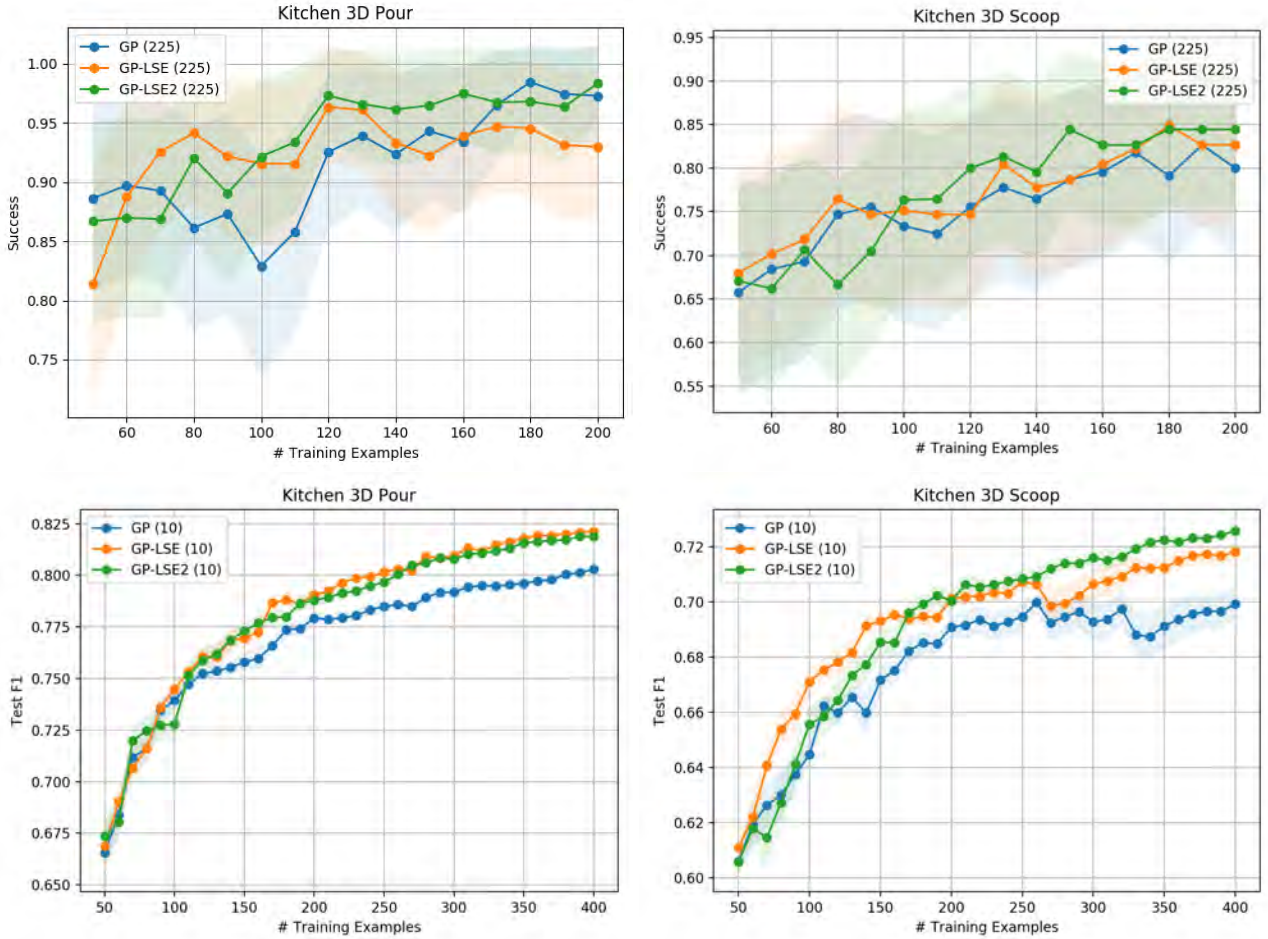
7.2 Active learning

We also evaluated the impact of training a GP using active learning on the success rate and F1 score learning curves in this setting. We compare a GP trained *without* active learning (GP) with two GPs trained *with* active learning strategies, both of which use the straddle algorithm (GP-LSE, GP-LSE2). Each GP uses the multi-layer perceptron kernel.

When actively training a model in the real world, the learner can fairly quickly evaluate any control parameter. However, this is not necessary true for context parameters because they often involve properties of physical objects. If we applied the straddle algorithm to perform a *continuous* optimization over context parameters, we would need to fabricate objects with the selected sizes in order to faithfully score the trial. While this could be possible through, for example, 3D printing, the real-world time and resource overhead would make it prohibitive. However, given a finite set of contexts derived from a fixed set of existing objects, it is possible to perform a *continuous* optimization over control parameters per *discrete* context parameter select the best parameter pairs. Still, this assumes that the robot can select the next context, which might not be true for a robot learning online in the wild.

Motivated by the semantic differences between context and control parameters, we experimented with three partitions of parameters into those that are sampled uniformly at random and those that are actively optimized in some manner. Specifically, we compared sampling all parameters (GP), actively optimizing all parameters (GP-LSE), and sampling the context parameters but actively optimizing the control parameters with respect to the context parameters (GP-LSE2).

In order to faithfully train an active learner, training must be performed in series because every new training example modifies the learner's posterior and thus influences the selection of the next trial. As a result, active learning must be performed serially while trials selected independently and

Figure 11. Pouring and scooping learning curves comparing a GP trained without active learning (GP), a GP that actively selects both the context and control parameters (GP-LSE), and a GP that only actively selects the control parameter (GP-LSE2). *Top row*: the success rate of the most confident control parameter produced by each learner. *Bottom row*: the test F1 score for each learner.

randomly can be collected massively in parallel. Because planning and simulating each trial takes at least 30 seconds, training several active learners over hundreds of training examples can be computationally burdensome. To expedite experimentation, we performed *discrete* active learning over the set of 10,000 training examples that we initially gathered randomly. The active learners score each example using the straddle acquisition function and extract the example with maximum value without replacement.

Figure 11 (*top*) displays the success rate of the three learners using the same experimental conditions as in section 7.1.1. The active learners (GP-LSE, GP-LSE2) outperform the non-active learner (GP). Ultimately, the active learner that randomly samples the context (GP-LSE2) resulted in the best success rate. Figure 11 (*bottom*) displays the F1 score of the three learners using the same experimental conditions as in section 7.1.2. Here, the active learners (GP-LSE, GP-LSE2) more conclusively outperform the non-active learner (GP). Achieving a high F1 score requires making accurate predictions for most of the decision space, not just a single point per context. As a result, methodically exploring high-variance regions outperforms random sampling.

Because gathering real-world data is labor intensive, we desired learning good pouring and scooping models with only around 100 training examples. Thus, we performed

a extensively-repeated experiment over a fewer samples in order to simulate our real-world experiments (described in section 8.3). Instead of training on 400 examples for 10 episodes, we trained on only 100 examples but for 200 episodes. Figure 12 displays the F1 score for this experiment. Although the variance is non-trivial, the active learner that randomly samples the context (GP-LSE2) results in the best average performance. Our hypothesis is that, because control parameters are often more predictive of the score than the context parameters, GP-LSE focuses its attention on reducing uncertainty along the control dimensions, possibly neglecting the context dimensions.

7.3 Adaptive and diverse sampling

Given a probabilistic estimate of a desirable set of θ values, obtained by a method such as GP-LSE, the next step is to sample values from that set to use in planning. We compare simple rejection sampling using a uniform proposal distribution (REJECTION), the basic adaptive sampler from section 4.2, and the diversity-aware sampler from section 4.4 with a fixed kernel: the results are shown in table 1. For all the results, we use $\Phi^{-1}(0.99\Phi(\beta_*))$ to construct the high probability super-level set.
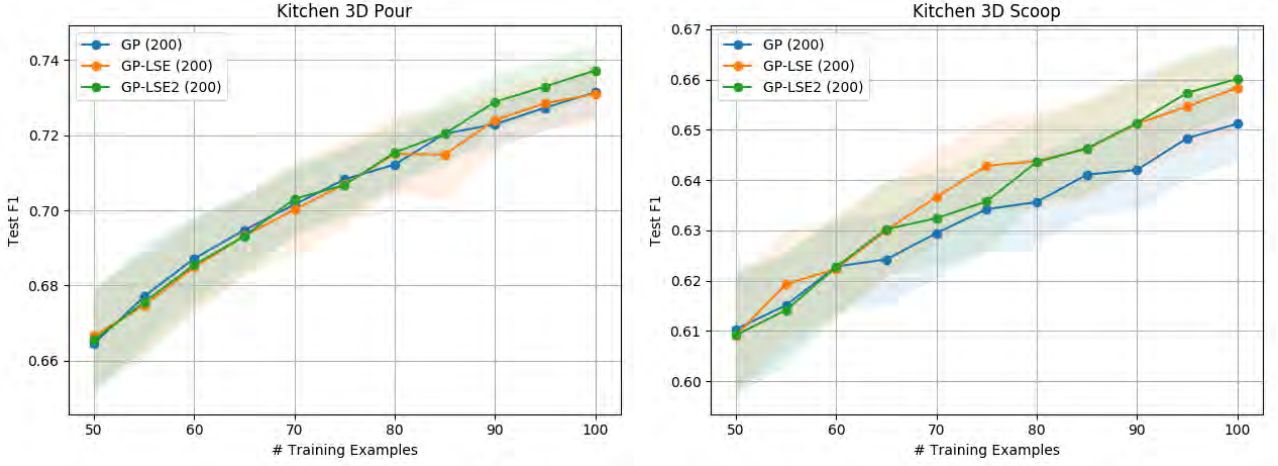
Figure 12. Pouring and scooping test F1-score learning curves from 50 to 100 training examples for the learners described in figure 11. Here, the learner process was repeated 200 times per learner in order to more accurately capture the variance in performance when using a small number of training examples.

Table 1. Effectiveness of adaptive and diverse sampling. FP: the false positive rate of 50 samples. T_{50}: the total sampling time of the 50 samples. N_5: number of samples required to achieve 5 positive ones. Diversity: the diversity rate of the 5 positive samples.

		REJECTION	ADAPTIVE	DIVERSE
Pour (3D)	FP (%) ↓	0.03 ± 0.10	0.02 ± 0.07	0.02 ± 0.08
	T_{50} (s) ↓	143.56 ± 176.05	72.84 ± 71.26	65.93 ± 72.93
	N_5 ↓	5.14 ± 0.45	5.10 ± 0.58	5.15 ± 0.71
	Diversity ↑	15.29 ± 3.44	15.40 ± 2.94	18.78 ± 3.07
Scoop (3D)	FP (%) ↓	0.13 ± 0.17	0.16 ± 0.16	0.12 ± 0.10
	T_{50} (s) ↓	265.57 ± 118.24	72.84 ± 71.26	35.11 ± 18.73
	N_5 ↓	5.77 ± 1.82	6.11 ± 1.77	5.66 ± 1.09
	Diversity ↑	10.93 ± 2.50	11.82 ± 1.63	14.57 ± 2.13

We report the false positive rate (FP)[‡] on 50 samples, the time to sample these 50 samples (T_{50}), the total number of samples required to find 5 positive samples (N_5), and the diversity of those 5 samples. The experiments are repeated over 50 such samplers for each method. We do not limit CPU time for gathering 50 samples for 3D simulated experiments. The diversity term is measured by $D(S) = \log \det(\Xi^S \zeta^{-2} + I)$ using a squared exponential kernel with inverse length scale $l = [1, 1, ..., 1]$ and $\zeta = 0.1$. We run the sampling algorithm for an additional 50 iterations (a maximum of 100 samples in total) until we have 5 positive examples and use these samples to report the diversity quantity $D(S)$.

DIVERSE uses slightly more samples than ADAPTIVE to achieve 5 positive ones, and its false positive rate is slightly higher than ADAPTIVE, but the diversity of the samples is notably higher. The FP rate of diverse can be decreased by increasing the confidence bound on the level set.

7.4 Learning kernels for diverse sampling

In the final set of experiments, we explore the effectiveness of the diverse sampling algorithm with task-level kernel learning. We compare ADAPTIVE, DIVERSE-GK with a fixed kernel and diverse sampling with learned kernel (DIVERSE-LK), in every case using a high-probability super-level set

Table 2. Effect of distance metric learning on sampling.

WASH	Runtime (s) ↓	60s SR (%) ↑	6s SR (%) ↑
ADAPTIVE	18.41 ± 8.87	42.0 ± 10.3	28.0 ± 15.4
DIVERSE-GK	18.22 ± 9.70	48.0 ± 7.5	26.0 ± 16.6
DIVERSE-LK	17.07 ± 9.72	53.0 ± 6.0	40.0 ± 11.8
UNCLOG	**Runtime (s) ↓**	**60s SR (%) ↑**	**6s SR (%) ↑**
ADAPTIVE	44.20 ± 22.05	23.0 ± 12.5	5.0 ± 3.2
DIVERSE-GK	44.85 ± 23.47	21.0 ± 9.2	5.0 ± 3.2
DIVERSE-LK	42.86 ± 23.34	23.0 ± 12.1	6.0 ± 5.8

estimated by a GP. All the experiments are repeated 5 times with random scene settings. In DIVERSE-LK, we use $\epsilon = 0.3$.

To test the performance of kernel learning, we design two tasks that require sophisticated manipulation to accomplish the goals. In the first task, called WASH, the goal is to pour (*e.g.* dish liquid) from a cup to a bowl while avoiding collisions with the faucet next to the bowl. The second task, called UNCLOG, aims to scoop (*e.g.* food waste) from a bowl-shaped sink while avoiding collisions with the faucet. We select a fixed test set with 50 task specifications and repeat the evaluation 5 times. Different task specifications have different faucet sizes, bowl shapes, spoon sizes, cup sizes, faucet heights and distances between faucet and bowl. Figure 13 shows examples for task WASH and task UNCLOG.

We show the timing and success rate results in table 2 (after training). Our empirical results shows that, in general, DIVERSE-LK is able to find a better solution than the alternatives in both of these tasks. This suggests that the kernel learning approach that we adopted is indeed generating more suitable samples for the planner.

7.5 Integrated system

Finally, we integrated the learned sampling models for the `pour` and `scoop` actions with 7 pre-existing robot operations (`move`, `pick`, `place`, `fill`, `push`, `stir`) in a domain specification for PDDLStream.

[‡] The proportion of samples that do not satisfy the true constraint.
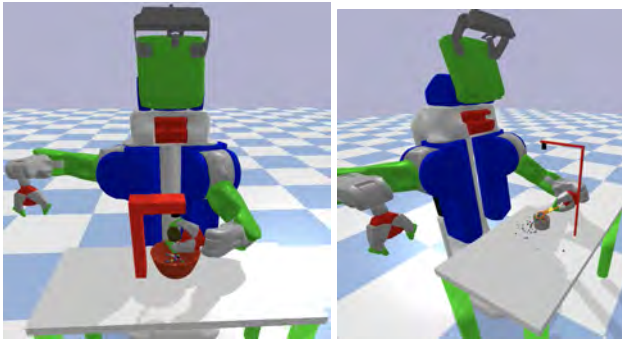
Figure 13. Examples for Task WASH and UNCLOG. Task WASH's goal is to pour from a cup to a bowl while avoiding the faucet next to the bowl. Task UNCLOG is to scoop from a bowl while avoiding the faucet next to the bowl.

Figure 14. *Left:* R-CNN table and object 2D visual bounding box detections. *Right:* the estimated table surface and object poses visualized in RViz. The table surface plane normal is the blue vector, the yellow rectangle is the axis-aligned bounding of the surface within the plane, and the blue polygon is the convex hull of the surface within the plane. The colored mesh of each registered object pose is overlaid on the point cloud, demonstrating the accuracy of the position and orientation estimates.

As a demonstration, we give the robot a goal which is to "prepare" a cup of coffee with cream and sugar. To achieve this goal, the robot must pour coffee into the white bowl, scoop sugar from the red bowl and dump it into the white bowl, and stir the while bowl, and return to its initial configuration. While doing this, the robot also needs to plan its path in a way that avoids all obstacles. Figure 1 displays the robot solving a *Kitchen3D* (*left*) and real-world (*right*) version of this task. See https://tinyurl.com/lis-ltamp for a video of a real-world robot solving this task.

These results illustrate the ability to augment the existing competences of a robotic system (such as moving while avoiding collisions) with new sensorimotor primitives by learning probabilistic models of their preconditions and effects and using a state-of-the-art domain-independent continuous-space planning algorithm to combine them fluidly and effectively to achieve complex goals.
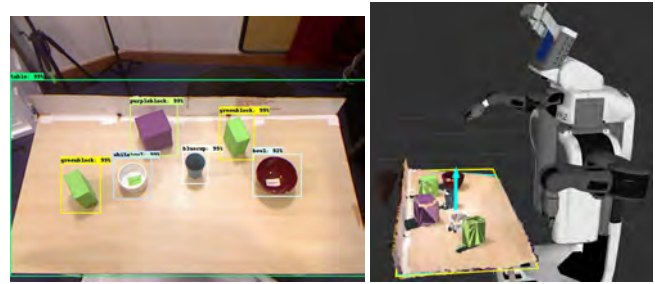
8 Real-world experiments

We applied our learning and planning framework to several real-world problems to demonstrate its sample efficiency and ability to generalize over a diverse set of planning scenarios. We use the same set of PyBullet primitive implementations as in the *Kitchen3D* simulation. An open-source implementation of system is available at https://github.com/caelan/LTAMP.

8.1 Perception

We assume that the objects rest on a single table and are fully observable from a Kinect RGB-D camera mounted on the robot's head. Additionally, we assume that we have an approximate 3D mesh model for each object on the table. We created crude mesh models for each bowl and cup, derived solely from rough base diameter, top diameter, and height measurements.

We use visual data to recognize and coarsely locate objects on the table and depth data to identify the table surface and localize objects. We use the Faster R-CNN (Ren et al. 2015) visual object detector (Huang et al. 2017) implemented in TensorFlow (Abadi et al. 2016) to predict labeled 2D bounding boxes for the table and each object model. We pretrained the R-CNN on the COCO data set (Lin et al. 2014) and then trained it on 447 hand-annotated image arrangements of our table, blocks, cups, and bowls. An example set of detections is displayed in figure 14 (*left*).

We use the detection information to isolate subsets of the point cloud contained within the 3D view cone corresponding to each 2D bounding box. For each detected table, we use the Point Cloud Library's (PCL) (Rusu and Cousins 2011) random sample consensus (RANSAC) (Fischler and Bolles 1981) plane estimator to obtain the equation of its plane as well as the 2D convex hull of its points when projected into the plane. We filter planes with normal vectors that significantly deviate from the global z-axis. Then, we prune detected objects that are not supported by the estimated table plane. For the remaining detected objects, we perform pose registration on the point cloud contained within its cone using the mesh model corresponding to the predicted label. We use a pose estimator built by Glover (2014), which performs a randomized optimization over object placements resting normal to the plane, minimizing the distance between the observed point cloud and a point cloud derived from the mesh. Figure 14 (*right*) shows the estimated table surface plane as well as the detected objects at their estimated poses.

Figure 15 provides a flowchart of our system. We use the Robot Operating System (ROS) (Quigley et al. 2009) to relay plane and pose estimates to the Python planning engine where they are treated as the ground-truth environment. The inputs are RGB, depth, and joint data as well as a goal description. The perception subsystem is used to populate an estimate of the initial state. The planning subsystem consumes this estimate along with the goal description, motion planning primitives, and the current Gaussian Process models. After receiving a single observation, the planner solves the corresponding problem and outputs a path that specifies a sequence of robot arm joint positions. After solving for a plan, the execution subsystem performs local feedback control to follow the plan, scores the final world state, and adds the result to the training data set. After interpolation using cubic splines, the resulting trajectory is executed in an entirely open-loop manner at the high level.
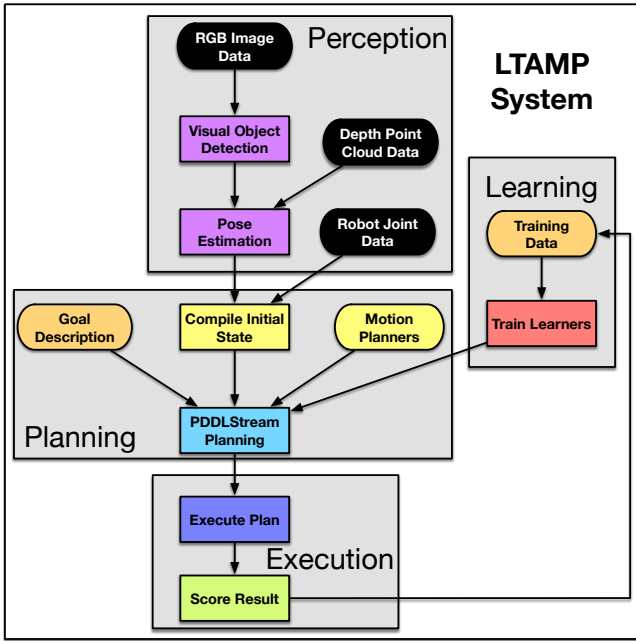
Figure 15. A flowchart that decomposes our real-world system into four components: perception, learning, planning, and execution.

Figure 16. *Left*: the training set of 10 bowls, 12 cups, and 3 spoons. *Right*: the testing set of 5 bowls, 6 cups, and the same 3 spoons.

8.2 Data collection

We use small bead-like objects as the material to be poured or scooped. Specifically, we use red wooden objects for pouring and dried chickpeas for scooping. In our training setup, we place bowls and cups on USB scales to estimate the particle mass contained within each object both before and after execution. The USB scales are directly connected to our computer to provide automated real-time mass readings. We subtract the mass of the bowl or cup in order to obtain the mass of the particle contained within an object.

In simulation, the world can be directly assigned to be the state prior to executing a skill. However, in the real world, the robot must also act to set up a skill (*e.g.* grasping the cup), act to score the skill (*e.g.* dumping the spoon's contents into a bowl), and reset the scene for its next trial. It is critical that the robot respects kinematic, joint-limit, collision, and spillage orientation constraints in order to ensure that these actions are likely to be successfully executed. We use our planner to plan paths that respect these constraints and facilitate data collection. Thus, we are performing both *learning for planning* and *planning for learning*.

We formulate collecting one trial as a planning problem where the planner is restricted to use a single control parameter that is selected either uniformly at random or by a GP active learner. Otherwise, the planner has the freedom to select the other plan parameters, such as the grasp used to pick the cup. For both pouring and scoring, we enforce that the cup finishes at its initial pose and that the robot finishes at its initial configuration. By planning to reset the scene, we avoid the need to teleoperate the robot or manually extract an object from the robot's gripper. Additionally, this prevents the robot arm from self-occluding the table during its next observation. As a result, the only manual actions that a human must perform are cleaning up spilled particles and swapping the placed objects that will be used on next trial.

For pouring, the robot picks up the cup, attempts to pour its contents into the bowl using the sampled control parameter, places the cup back at its initial pose, and returns to the initial configuration. This results in the following sequence of operators: [move, pick, move, pour, move, place, move]. The fraction of particles that were successfully poured is the ratio of the final bowl particle mass to the initial cup particle mass. For scooping, the spoon starts in the robot's gripper, at an approximate grasp. The robot scoops the contents of the bowl using the sampled control parameter, dumps the spoon's contents into the measurement bowl, and returns to the initial configuration. This results in the following sequence of skills: [move, scoop, move, pour, move]. The fraction of particles that were successfully scooped is the ratio of the final measurement bowl particle mass to the mass capacity of the spoon, which is measured offline. As a result, only one scale is required when scoring a scoop. Finally, we use the plan constraint compilation procedure of Garrett et al. (2020b) to enforce that each plan exactly executes the prescribed sequence of skills, preventing it from considering plans that, for example, perform two scoops. Ultimately, the planner is typically able to find a solution in less than 15 seconds. See the "Learning to {Pour, Scoop}: Data Collection" videos at https://tinyurl.com/lis-ltamp for demonstrations of the robot collecting data using this pipeline.

We trained the learners on a set of training objects and evaluated the learners on a set of unseen testing objects. Several of the bowls and cups are from the YCB dataset (Calli et al. 2015). The objects range in both size, mass, and material (ceramic, plastic, and 3D printed). We trained the learners on 10 bowls and 12 cups of varying sizes. We tested the learners on 5 bowls and 6 cups of varying sizes. We used the same set of 3 spoons both during training and testing. The number of pouring contexts is the number of bowl and cup pairs while the number of scooping contexts is the number of bowl and spoon pairs. Figure 16 displays the set of training and testing objects. For each trial, we sampled the objects (and as a result the context) uniformly at random.

To quickly collect data incorporating new objects without needing to retrain the R-CNN, we developed a user interface (UI) that allows a user to "replace" the object detector by manually annotating object bounding boxes online. These labeled bounding boxes are then sent to the point-cloud pose-estimation system as normal. The labels of each bounding box can also be changed programmatically, enabling the
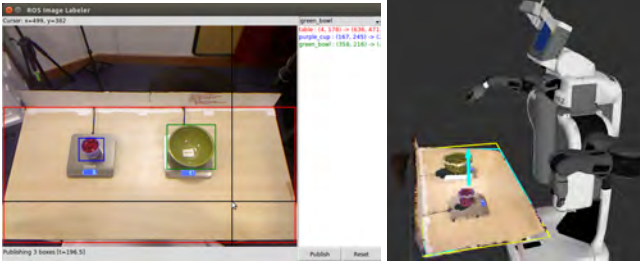
Figure 17. *Left:* the manual bounding box labeling tool that replaces the R-CNN predictions during training. *Right:* the corresponding estimated table surface and object poses visualized in RViz. See figure 14 (*right*) for a description of the RViz markers.

data collection program to update their values given the next selected cup and bowl pair. Figure 17 displays the UI tool and visualizes the corresponding table plane, registered bowl mesh, and registered cup mesh for a pouring trial.

8.3 Training

We compared the sample efficiency of GPs trained both with and without active learning in this real-world setting. Both GPs used the MLP kernel as well as the parameterization in section 6.2. We initially seeded each learner with 50 training examples gathered by sampling context and control parameters uniformly at random.

Figure 18. The robot executing actively selected control parameters. The learners intentionally explore control parameters that are near the boundary of the super-level set. *Left*: the selected pour successfully produces several particles in the bowl but also spills many particles. *Right*: the selected scoop is able to scoop some particles, but the spoon still has the capacity to hold more.

Figure 18 visualizes the robot executing pour and scoop actions selected using active learning. Both of these trials demonstrate borderline success, which is consistent with the robot selecting control parameters near the zero level set. Figure 19 visualizes selected pours and their scores overlaid on a particular bowl in our *Kitchen3D* simulator. Red cups indicate pours with negative scores and blue cups indicate pours with positive scores. The three images compare selections made uniformly at random, by the GP active learner, and by the final trained GP learner on test objects. Many of the active learner's selections are green, indicating that they are near the zero level set boundary.

During training, we tested how well the learners were able to *classify* successful pours and scoops. This allowed us to obtain an measure of how well the GP was learning without needing to periodically evaluate on testing data during training. We collected a test data set of 133 pour
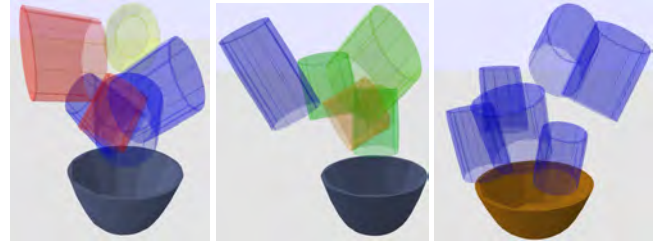
Figure 19. The distribution of selected real-world pours visualized in simulation per selection policy. The measured score of the pour is visualized by the hue of the cup, where red pours are the least successful, green pours are near the zero level-set, and blue pours are most successful. *Left*: pours selected *uniformly at random* for a single training bowl. *Center*: pours selected *actively* for the same training bowl. Many selected pours are green, indicating that the learner is exploring the decision boundary. *Right*: the most confident pours for a single testing bowl. Each pour is blue, which indicates that all pours were successful.

and 81 scoop examples, sampled uniformly at random on the test objects. Figure 20 displays the F1-score learning curves of the GP learners without and with active learning on this data set. The $1/4$ standard deviation error bounds result from retraining each GP 10 times on the *same* data, to account for the stochastic hyper-parameter optimization when retraining. Active learning enables the GP learner to more quickly classify successful pours and scoops.

8.4 Most confident prediction

Recall that our ultimate objective is to sample control parameters that the learner confidently believes to lie in the super-level set. We compared the most confident predictions of a GP trained on 50 training examples sampled uniformly at random, 100 (96 for scooping) training examples sampled uniformly at random, and 50 training examples sampled uniformly at random followed by 50 (46 for scooping) selected actively. We performed one trial per unique bowl-cup and bowl-spoon test pair, resulting in 30 pours per learner and 15 scoops per learner.

Table 3 lists the performance of each learner when making its most confident prediction on the test objects. *Valid* is the percentage of sampled control parameters for which full motions of the robot could be found. Recall that the learner may predict control parameters that cannot be safely executed by the robot, such as pours in the interior of a bowl that do not admit any collision-free grasps. *Success* is the percentage of sampled control parameters that were in the super-level set. *Filled* is the percentage of the cup or spoon's capacity was filled. The active learner outperforms the non-active learners in each metric both for pouring and scooping.

8.5 Integration

Finally, we used our learned pouring model within our planner to solve challenging real-world multi-step manipulation problems. We experimented with two problems where the robot must combine its learned pouring models with motion planners that respect kinematic and collision constraints. In each problem, the blue cup is initially holding "liquid" particles, and the goal is for the brown bowl to
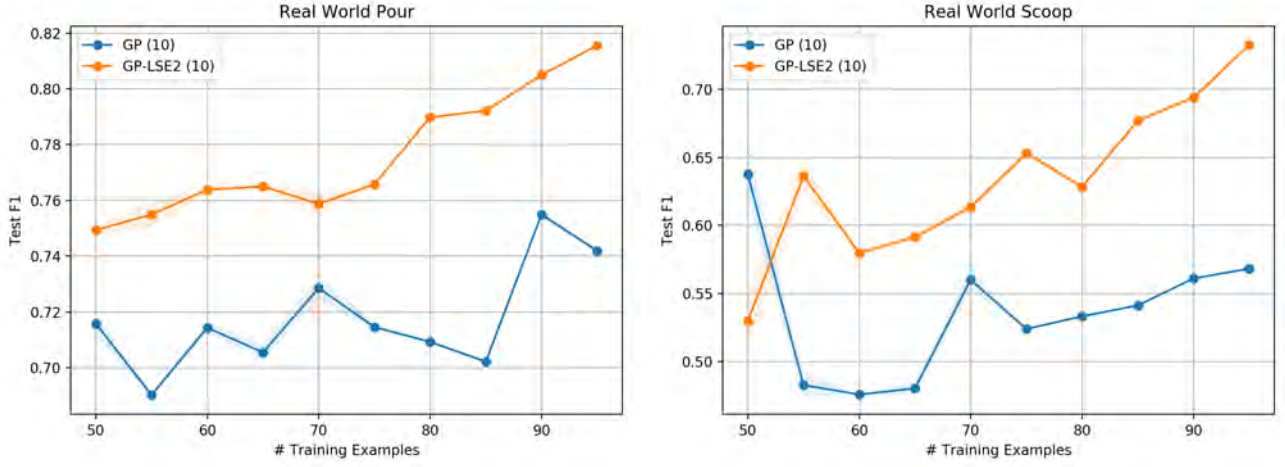
Figure 20. The F1 score for a GP trained both *without* (GP) and *with* active learning (GP-LSE2) on the testing objects. Each GP was initially trained with 50 examples collected uniformly at random on training objects.

		Batch N_{50}	Batch N_{100}/N_{96}	Active N_{100}/N_{96}
Pour	Valid (%)	0.867	0.833	0.933
	Success (%)	0.923	0.920	1.000
	Filled (%)	0.964	0.958	0.994
Scoop	Valid (%)	0.600	0.933	0.933
	Success (%)	0.889	0.786	0.929
	Filled (%)	0.829	0.861	0.952

Table 3. When evaluating pours and scoops during testing, the percentage of them that admitted a full robot plan (*valid*), the percentage of them that were in the super-level set (*successful*), and the average mass inside the scoring bowl relative to the capacity of the involved cup or spoon. The best value of each metric across the three learners is indicated in red.

instead contain the particles. The robot must additionally return to its initial configuration with both grippers empty.

Figure 21 demonstrates the robot solving the first problem. In this problem, the robot is unable to find a kinematically feasible way of picking the blue cup without colliding with the green block. Thus, the robot plans to pick the green block and finds a placement for it that allows for the blue cup to be picked. Afterwards, the robot can now safely pick the blue cup and pour its contents into the brown bowl. Finally, the robot places the blue cup and moves its left arm back to its initial configuration. Critically, the robot finds a grasp for the cup that both admits a pour path that is predicted to be successful and admits a collision-free pick path when the green block is moved. See `https://youtu.be/a5F1hce4o0o` for a video of the robot executing this solution.

Figure 22 demonstrates the robot solving the second problem. In this problem, the bowl starts at one side of the table while the blue cup starts on the other side. Because neither arm can reach both objects, the robot must intentionally manipulate one of the objects with one arm to put it within reach other the other arm. There are two high-level ways of accomplishing this. The first requires picking up the blue cup with the robot's left arm and deliberately placing it near the middle of the table, within reach of the right arm. The second requires pushing the brown bowl with its right arm towards the middle of the table. Although the robot's planning model can produce both solutions, the

planner returned the second solution, likely because it uses fewer actions. Once the brown bowl is within reach, the robot can successfully pour the contents of the blue cup into the bowl and return to its initial state. Because the robot was initially kinematically unable to pour using its left arm, it intentionally identifies a pose that it can push the bowl to in order to be within reach. See `https://youtu.be/a5F1hce4o0o?t=43` for a video of the robot executing this solution.

9 Conclusion

This paper addresses learning generative models for dynamic manipulation skills for use during multi-step manipulation planning. We learn the conditions for which a pour or scoop manipulation skill is sufficiently successful using Gaussian processes. This allows us to capture the uncertainty in the learner's model, enabling us to make risk-aware predictions and perform active learning to methodically select training examples that best reduce the model's uncertainty. Through simulated and real-world experiments, we show that active learning reduces the number of robot trials required to learn a skill. Additionally, we introduce methods for diversely exploring the set of successful pours or scoops. This enables a planner to quickly find values that admit a full robot plan. Our integrated planner combines learned models for pouring and scoping with conventional robotics operations, enabling it to generalize across a large set of challenging manipulation problems.

9.1 Future work

One important avenue for future work involves incorporating learner predictions into the action *cost* of the associated control parameter. Costs could be derived from the expected score of the parameter or the probability that the parameter is in the super-level set. There are several approximate methods for performing risk-aware deterministic planning with non-negative additive costs (Garrett et al. 2020b). This would enable the planner to weigh the expected cost of executing a sequence of control parameters among several candidate plans.
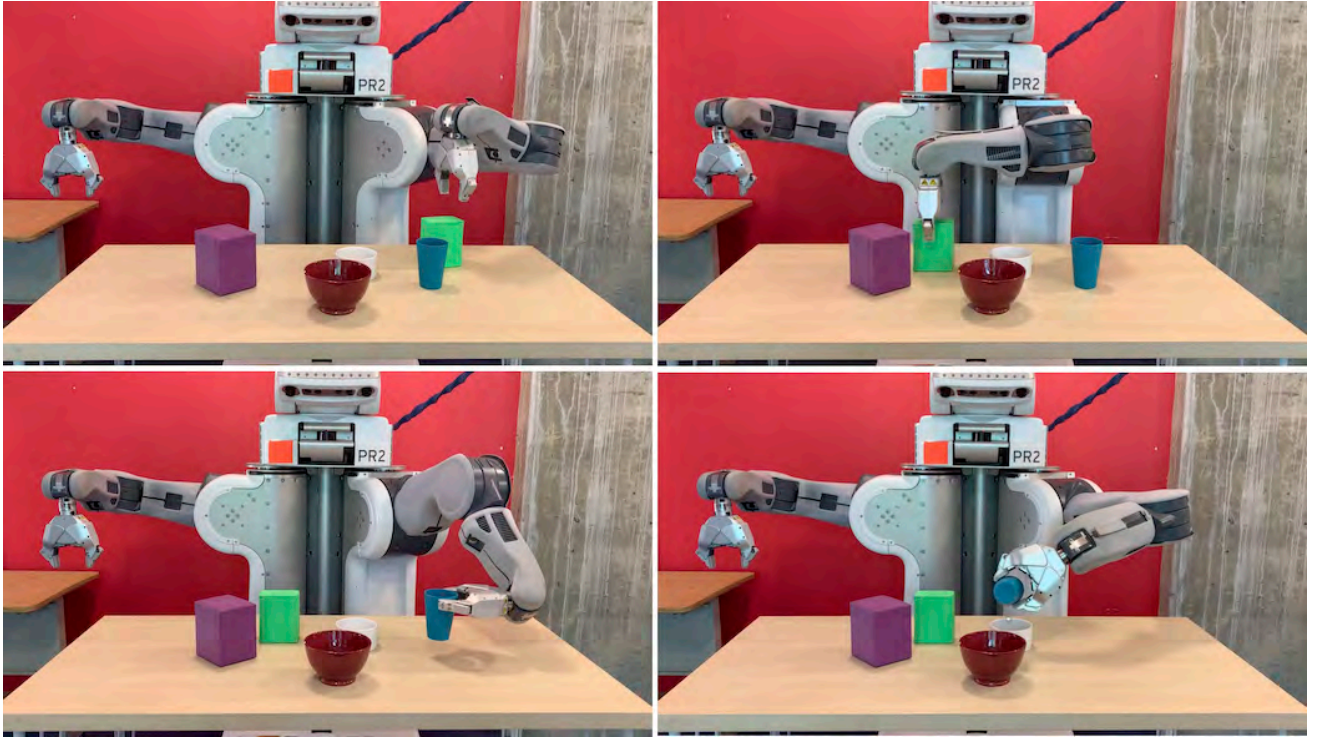
Figure 21. The goal is for the particles in the blue cup to be in the white bowl. Because the green block obstructs reachable side grasps for the blue cup, the planner automatically plans to relocate the green block before picking the blue cup and pouring its contents into the white bowl. From *left-to-right* and *top-to-bottom*, the robot picking the green block, the robot placing the green block, the robot picking the blue cup, and the robot pouring the blue cup's contents into the brown bowl.
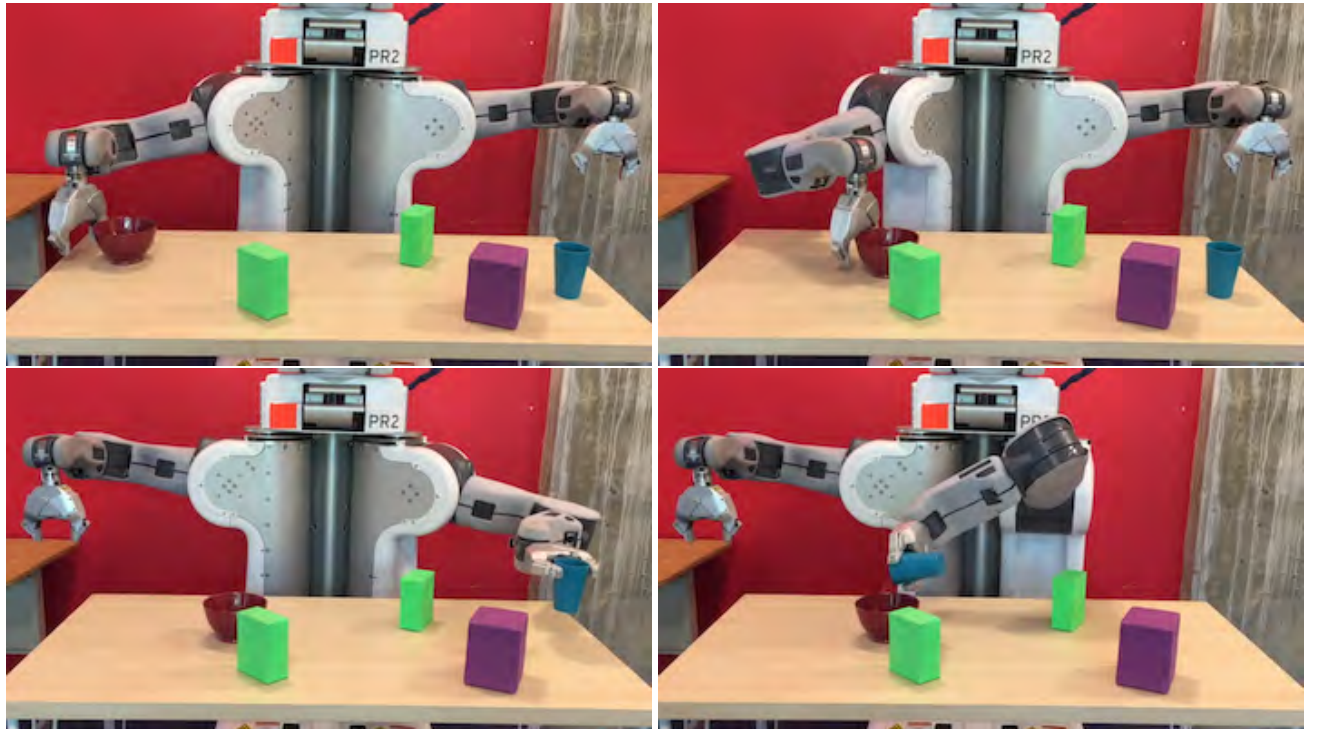
Figure 22. The goal is for the particles in the blue cup to be in the brown bowl. Because the robot cannot reach the brown bowl with its left arm, the planner automatically plans to push the bowl towards the center of the table, so it can then pour the blue cup's contents into the brown bowl. From *left-to-right* and *top-to-bottom*, the state before the robot pushes the brown bowl, the resulting state after the push, the robot picking the blue cup, and the robot pouring the blue cup's contents into the brown bowl.

Although we consider both learning in simulation and the real world, we have not addressed sim-to-real transfer; which may be useful in settings where a high-fidelity simulator is available. In our preliminary investigation, we found that it was challenging to benefit from active learning when training real-world models on simulated data. Intuitively, if simulation and the real-world mismatch, particularly with respect to which dimensions are most informative, the

active learner may explore training examples that do not effectively decrease uncertainty in the model. Ultimately our goal is to develop methods that can learn effectively from a few real-world samples, without the need to develop a simulation. However, in settings where a simulator exists, further investigation of the effective integration of active learning and sim-to-real transfer is desirable.

Finally, this paper addresses deterministic planning and open-loop execution; however, the real-world is stochastic and partially observable. Our current and future work involves learning models for stochastic manipulation actions and observation actions for use during belief-space planning (Garrett et al. 2020b; Kaelbling and Lozano-Pérez 2013).

Acknowledgements

A   Gaussian processes

Gaussian processes (GPs) represent distributions over functions and serve as a useful representation for Bayesian regression. In a GP, any finite set of function values has a multivariate Gaussian distribution. We use $GP(\mu, k)$ to denote a GP with mean function $\mu(\boldsymbol{x})$ and kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$. Two frequently used stationary covariance kernel functions are the *squared exponential* and *Matérn kernels*. Let $r = (\boldsymbol{x} - \boldsymbol{x}')^\top (\boldsymbol{x} - \boldsymbol{x}')$. Then the squared exponential kernel is

$$k_f(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \mathrm{e}^{-\frac{1}{2\ell_f^2} r},$$

with a variance σ_f^2 and length scale hyper-parameter ℓ_f. The Matérn kernel is given by

$$k_m(\boldsymbol{x}, \boldsymbol{x}') = \sigma_m^2 \frac{2^{1-\xi}}{\Gamma(\xi)} \Big(\frac{\sqrt{2\xi r}}{h}\Big)^\xi B_\xi \Big(\frac{\sqrt{2\xi r}}{h}\Big),$$

where Γ is the gamma function, B_ξ is a modified Bessel function. Its hyper-parameters are σ_m^2, l_m and a roughness parameter ξ. Additionally, we consider the non-stationary *multi-layer perceptron kernel* (also called the neural network kernel) (Neal 2012; Rasmussen and Williams 2006), which often better models discontinuous functions such as the score functions in Section 6.1 (Vasudevan et al. 2009; O'Callaghan and Ramos 2012),

$$k_n(\boldsymbol{x}, \boldsymbol{x}') = \frac{2\sigma_n^2}{\pi} \sin^{-1} \frac{\tilde{\boldsymbol{x}}^\top \boldsymbol{\Sigma}^2 \tilde{\boldsymbol{x}}'}{\sqrt{\tilde{\boldsymbol{x}}^\top \boldsymbol{\Sigma}^2 \tilde{\boldsymbol{x}} + 1} \sqrt{\tilde{\boldsymbol{x}}'^\top \boldsymbol{\Sigma}^2 \tilde{\boldsymbol{x}}' + 1}},$$

where $\tilde{\boldsymbol{x}} = [1, \boldsymbol{x}]$. Its hyper-parameters are a diagonal covariance matrix $\boldsymbol{\Sigma}^2$ and variance σ_n^2.

Let f be a true underlying function sampled from $GP(0, k)$. Given a set of observations $\mathcal{D} = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^{|\mathcal{D}|}$, where y_t is an evaluation of f at $\boldsymbol{x}_t$ corrupted by i.i.d additive Gaussian noise $\mathcal{N}(0, \zeta^2)$, we obtain a posterior GP, with mean

$$\mu(\boldsymbol{x}) = \boldsymbol{k}^\mathcal{D}(\boldsymbol{x})^\mathrm{T} (\boldsymbol{K}^\mathcal{D} + \zeta^2 \boldsymbol{I})^{-1} \boldsymbol{y}^\mathcal{D}$$

and covariance

$$\sigma^2(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}^\mathcal{D}(\boldsymbol{x})^\mathrm{T} (\boldsymbol{K}^\mathcal{D} + \zeta^2 \boldsymbol{I})^{-1} \boldsymbol{k}^\mathcal{D}(\boldsymbol{x}')$$

where the kernel matrix $\boldsymbol{K}^\mathcal{D} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{D}}$ and $\boldsymbol{k}^\mathcal{D}(\boldsymbol{x}) = [k(\boldsymbol{x}_i, \boldsymbol{x})]_{\boldsymbol{x}_i \in \mathcal{D}}$ (Rasmussen and Williams 2006). With slight abuse of notation, we denote the posterior variance by $\sigma^2(\boldsymbol{x}) = \sigma^2(\boldsymbol{x}, \boldsymbol{x})$, and the posterior GP by $GP(\mu, \sigma)$. We use GPy (2012) for GP training with Auto Relevance Determination (ARD) (Wipf and Nagarajan 2008) to optimize for kernel hyper-parameters that maximize the likelihood of the data.

B   PDDLStream for TAMP

PDDLStream (Garrett et al. 2020a) is a framework for planning in the presence of sampling procedures. The use of sampling procedures enables PDDLStream to address hybrid discrete-continuous planning domains, such as TAMP domains. PDDLStream extends Planning Domain Definition Language (PDDL) (McDermott et al. 1998) by adding *streams*, declarative specifications of conditional samplers. Streams have previously been implemented by a human engineer through leveraging collision checkers, inverse kinematic solvers, and off-the-shelf motion planners. In this work, we learn new conditional samplers for dynamic skills, such as pouring and scooping skills, which are difficult for a human to correctly specify. An open-source implementation of PDDLStream is available at: `https://github.com/caelan/pddlstream`.

B.1   PDDL

In PDDL, states consist of a set of true *facts*, which are equivalent to parameterized Boolean variables. Actions (`:action`) are defined by a set of free *parameters* (`:param`), a *precondition* logical formula of facts (`:pre`) that must hold in a state in order to apply the action, and an *effect* logical conjunction of facts (`:eff`) that specifies which facts are set to true or false after applying the action.

Figure 23 gives the PDDL description of two of the actions that we consider: `pour` and `scoop`. These actions use the parameters `?arm`, `?bowl`, `?cup`, `?spoon`, and `?particles` to refer to a robot arm, bowl, cup, spoon, and set of "liquid" particles. Additionally, departing from typical PDDL models, several of our parameters are multi-dimensional *continuous* values: `?pose` is a stable object placement in SE(3); `?grasp` is a rigid gripper grasp of an object in SE(3); `?conf` is a robot arm configuration (set of d joint angles) in R^d; `?obj-path` is an object path consisting of a sequence of poses; and `?arm-path` is a robot arm path consisting of a sequence of configurations.

The `pour` action can be applied if `?cup` initially contains `?particles`. After execution, `?bowl` now contains `?particles` *instead* of `?cup` as successful pours transfer the full contents of `?cup` into `?bowl`.

The `?scoop` action can be applied if `?bowl` initially contains `?particles`. After execution, `?spoon` now *also* contains `?particles`. Critically, `pour` and `scoop` have `GoodPour`, `GoodScoop`, and `Motion` preconditions defined on their parameter values. The `GoodPour` and `GoodScoop` conditions enforce that the action parameter values correspond to pours and scoops that are likely-to-be successful. The objective for sampling and thus learning is to produce parameter values that satisfy these constraints. The `Motion` fact relates the path of the robot arm to the path of a grasped cup or spoon. See Garrett et al. (2020a) for descriptions of `move`, `pick`, and `place` actions that are representative of the similar actions used in this work.

B.2 Streams

Streams are the key extension of PDDL that enable planning for high-dimensional, continuous systems. Streams have a procedural and a declarative component. The procedural component is a function from a set of input values to a sampler that generates a sequence of output values. This procedure is implemented using a programming language, such as Python. The declarative component specifies conditions on legal input values as well as properties that any generated output values are guaranteed to satisfy. Its syntax similar to that of actions: the *input parameters* (`:inp`) and the *output parameters* (`:out`) specify the number and names of streams inputs and outputs. The *domain* keyword (`:dom`) specifies a logical formula of "typing" facts that `:inp` values must satisfy in order to be legal inputs to the sampler. The *certified* keyword (`:cert`) asserts a logical conjunction of facts that pairs of `:inp` and `:out` values *always* satisfy.

Figure 24 gives the PDDLStream description of the `sample-pour-path`, `sample-scoop-path`, and `follow-obj-path` streams, which sample values that serve as inputs to the `pour` and `scoop` actions. The `sample-pour-path` and `sample-scoop-path` streams take in as inputs a `?bowl` at a specific `?pose` as well as a `?cup` or `?spoon`. They output `?cup` or `?spoon` paths sampled from the set of paths that are predicted to be within the super-level set of pours or scoops. These streams plan for a manipulated object before considering the robot at all. The `sample-obj-path` stream takes in as inputs an `?arm`, object `?obj` held at `?grasp`, and a desired path the object should follow. It outputs robot arm paths such that `?obj` follows `?obj-path` when at `grasp`. The specification these three streams modularly separates primitive sampling operations that are solvable using traditional model-based algorithms, such as motion planning, from those that are better addressed using learning. As a result, our planning approach retains the generalization and theoretical benefits of model-based approaches while also exhibiting the flexibility of learning methods when primitive models are not known. See Garrett et al. (2020a) for descriptions of streams that sample object placements, object grasps, and robot transit motions.

Figure 25 demonstrates how the `sample-pour-path` and `follow-obj-path` streams compose to ultimately produce robot pouring paths for control parameters values sampled by the pour GP. The `sample-pour-path` stream takes in the model of a bowl and cup and featurizes the models using their dimensions, producing a context parameter for the GP learner. The GP samples a control parameter, which specifies waypoints for the cup. We linearly interpolate through these waypoints to produce a full path for the cup, which is the output of the `sample-pour-path` stream. The `follow-obj-path` stream takes in the model of the static environment, a model of the robot, and tool paths produced by `sample-pour-path`, `sample-scoop-path`, or another stream. Using Cartesian trajectory tracking, it solves for a robot path that follows the tool path for a particular grasp. The PDDLStream planner instantiates the `pour` action using these paths and solves for a plan that uses these and other actions to achieve the goal.

B.3 Algorithms

PDDLStream problems consist of an initial state, goal state, set of actions, and set of streams. PDDLStream algorithms are *domain-independent*, meaning that they are able to solve PDDLStream problems without any additional problem information. The simplest PDDLStream algorithm, the INCREMENTAL algorithm (Garrett et al. 2017a,b), iteratively alternates between a sampling and a searching phase. During its sampling phase, it passes all legal combinations of input values to each stream and attempts to sample new output values. During its searching phase, it performs a discrete search, such as a breadth-first search, on the discretized state space resulting from the finite set of currently sampled values. If the discrete search finds a solution, INCREMENTAL terminates. Otherwise, this process repeats. More advanced algorithms can also be applied using the exact same PDDLStream problem description. For example, the FOCUSED algorithm (Garrett et al. 2017a,b) first searches over *plan skeletons*, plans with free parameters, before attempting to sample values for the parameters. This allows FOCUSED to more intelligently identify which samplers are relevant for solving the task.

C Simulated *Kitchen2D* domain

In the earlier version of this work (Wang et al. 2018), we built a simulated 2D kitchen environment, *Kitchen2D*, based on the physics engine Box2D (Catto 2011). For completeness, we include descriptions of *Kitchen2D* and its corresponding empirical results for the algorithms in section 3 and section 4.

In *Kitchen2D*, we build in the policies for different skills, *e.g.* pouring, scooping, pushing, and demonstrate that it is sample-efficient to learn models of additional skills. Once we obtain those learned models, sampling-based task and motion planners like PDDLStream can make use of them in an effective way to plan efficiently.

C.1 Implementation of Kitchen2D

Figure 26 shows several scenes indicating the variability of arrangements of objects in the domain. The parameterized actions are: moving the robot (a simple "free-flying" hand), picking up an object, placing an object, pushing an object, filling a cup from a faucet, pouring a material out of a cup, scooping material into a spoon, and dumping material from a spoon. The gripper has 3 general movement degrees

```
(:action pour
 :param (?arm ?bowl ?pose ?cup ?cup-path ?particles ?grasp ?conf1 ?conf2 ?arm-path)
 :pre (and (GoodPour ?bowl ?pose ?cup ?cup-path)
           (Motion ?arm ?cup ?grasp ?cup-path ?conf1 ?conf2 ?arm-path)
           (Particles ?particles) (HasParticles ?cup ?particles)
           (AtPose ?bowl ?pose) (AtGrasp ?arm ?cup ?grasp) (AtConf ?arm ?conf1)
           (not (UnsafePath ?arm ?arm-path)))
 :eff (and (AtConf ?arm ?conf2) (HasParticles ?bowl ?particles)
           (not (AtConf ?arm ?conf1)) (not (HasParticles ?cup ?particles))))


(:action scoop
 :param (?arm ?bowl ?pose ?spoon ?spoon-path ?particles ?grasp ?conf1 ?conf2 ?control)
 :pre (and (GoodScoop ?bowl ?pose ?spoon ?spoon-path)
           (Motion ?arm ?spoon ?grasp ?spoon-path ?conf1 ?conf2 ?arm-path)
           (Particles ?particles) (HasParticles ?bowl ?particles)
           (AtPose ?bowl ?pose) (AtGrasp ?arm ?spoon ?grasp) (AtConf ?arm ?conf1)
           (not (UnsafeControl ?arm ?control)))
 :eff (and (AtConf ?arm ?conf2) (HasParticles ?spoon ?particles)
           (not (AtConf ?arm ?conf1))))
```

Figure 23. The description of the `pour` and `scoop` actions. The underlined preconditions highlight facts that are certified by the GP learners.

```
(:stream sample-pour-path
 :inp (?bowl ?pose ?cup)
 :dom (and (Bowl ?bowl) (Pose ?bowl ?pose) (Cup ?cup))
 :out (?cup-path)
 :cert (and (GoodPour ?bowl ?pose ?cup ?cup-path) (ObjPath ?cup ?cup-path)))

(:stream sample-scoop-path
 :inp (?bowl ?pose ?spoon)
 :dom (and (Bowl ?bowl) (Pose ?bowl ?pose) (Spoon ?spoon))
 :out (?spoon-path)
 :cert (and (GoodScoop ?bowl ?pose ?spoon ?spoon-path) (ObjPath ?spoon ?spoon-path)))

(:stream follow-obj-path
 :inp (?arm ?obj ?grasp ?obj-path)
 :dom (and (Arm ?arm) (Grasp ?obj ?grasp) (ObjPath ?obj ?obj-path))
 :out (?conf1 ?conf2 ?arm-path)
 :cert (and (Motion ?arm ?obj ?grasp ?obj-path ?conf1 ?conf2 ?arm-path)
            (Conf ?arm ?conf1) (Conf ?arm ?conf2) (ArmPath ?arm ?arm-path)))
```

Figure 24. The descriptions of the `sample-pour-path`, `sample-scoop-path`, and `follow-obj-path` streams, which certify the `GoodPour` predicate, `GoodScoop` predicate, and `Motion` predicates respectively.
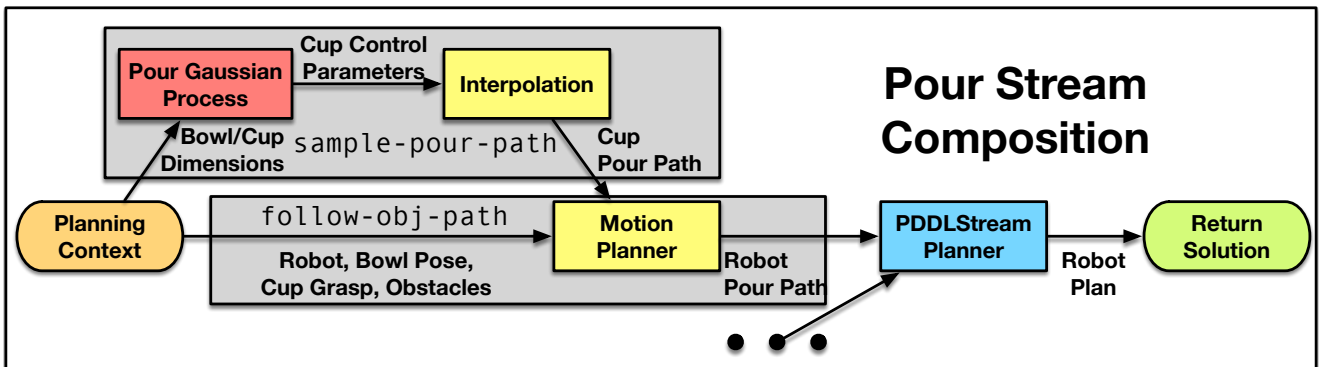
**Figure 25.** A flowchart that visualizes how the GPs connect to the `sample-pour-path` and `follow-obj-path` streams, which certify facts present in `pour` and `move` action preconditions.

of freedom (2D position and rotation) and can also open and close its fingers. The material to be poured or scooped is simulated as small circular particles. We use RRT-Connect (Kuffner Jr. and LaValle 2000) to plan motions for the gripper.
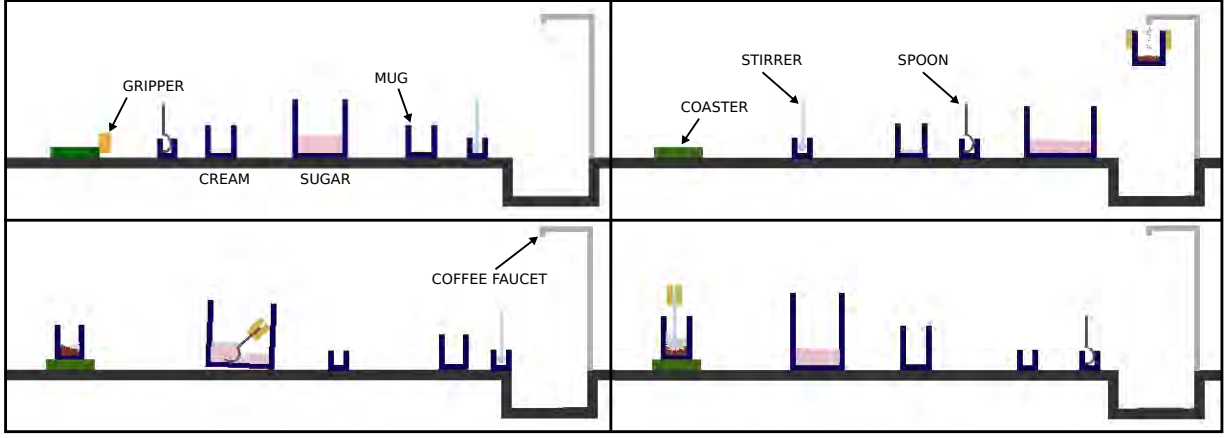
Figure 26. Four arrangements of objects in 2D kitchen, including: green coaster, coffee faucet, yellow robot grippers, sugar scoop, stirrer, coffee mug, small cup with cream, and larger container with pink sugar.
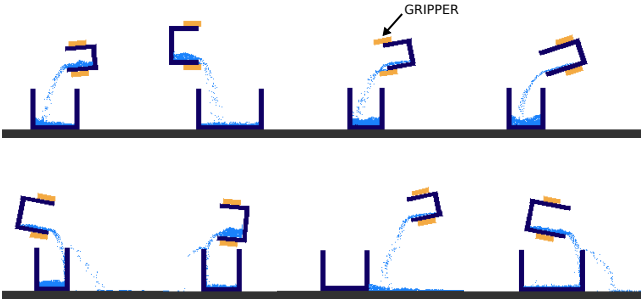
Figure 27. Examples of a gripper executing a pouring primitive in *Kitchen2D* for several contexts (cup dimensions) and control parameters (relative cup poses).

We learn models and samplers for three of these action primitives: pouring (4 context parameters, 4 predicted parameters, scooping (2 context parameters, 7 predicted parameters), and pushing (2 context parameters, 6 predicted parameters). The robot executes trajectories consisting of sequences of waypoints for the gripper, relative to the object it is interacting with.

As an example, figure 27 illustrates several instances of a parameterized sensorimotor policy for pouring in *Kitchen2D*. The skill has control parameters θ that govern the rate at which the cup is tipped and target velocity of the poured material. In addition, several properties of the situation in which the pouring occurs are very relevant for its success: robot configuration c_R, pouring cup pose and size p_A, s_A, and target cup pose and size p_B, s_B. To model the effects of the action we need to specify c'_R and p'_A, the resulting robot configuration and pose of the pouring cup A. Only for some settings of the parameters is the action feasible (*i.e.* $\chi(c_R, p_A, s_A, p_B, s_B, \theta, c'_R, p'_A) = 1$): the objective of our work is to efficiently learn a representation of the feasible region χ so as to enable a TAMP planner to use the skill, in conjunction with other skills, to solve novel problems.

For pouring, we use the scoring function $g_{pour}(x) = \exp(2(10x - 9.5)) - 1$, where x is the proportion of the liquid particles that are poured into the target cup. The constraint $g_{pour}(x) > 0$ means at least 95% of the particles are poured correctly to the target cup. The context of pouring includes the sizes of the cups, with widths ranging from 3

to 8 (units in Box2D), and heights ranging from 3 to 5. For scooping, we use the proportion of the capacity of the scoop that is filled with liquid particles, and the scoring function is $g_{scoop}(x) = x - 0.5$, where x is the proportion of the spoon filled with particles. We fix the size of the spoon and learn the action parameters for different cup sizes, with width ranging from 5 to 10 and height ranging from 4 to 8. For pushing, the scoring function is $g_{push}(x) = 2 - \|x - x_{goal}\|$ where x is the position of the pushed object after the pushing action and x_{goal} is the goal position; here the goal position is the context. The pushing action learned in section C.2 has the same setting as Kaelbling and Lozano-Perez (2017), viewing the gripper and object with a bird-eye view. The code for the simulation and learning methods is public at `https://ziw.mit.edu/projects/kitchen2d/`.

C.2 Experiments in Kitchen2D

Similar to our experiments in *Kitchen3D*, we show the effectiveness and efficiency of each component of our method independently, and then demonstrate their collective performance in the context of planning for long-horizon tasks in a high-dimensional continuous domain.

C.2.1 Active learning: We first demonstrate the performance of using a GP with the straddle algorithm (GP-LSE) to estimate the level set of the constraints on parameters for pushing, pouring and scooping in *Kitchen2D*. For comparison, we also implemented a simple method (Kaelbling and Lozano-Perez 2017), which uses a neural network to map (θ, α) pairs to predict the probability of success using a logistic output. Given a partially trained network and a context α, the $\theta^* = \arg\max_\theta \text{NN}(\alpha, \theta)$ which has the highest probability of success with α is chosen for execution. Its success or failure is observed, and then the network is retrained with this added data point. This method is called NN_c in the results. In addition, we implemented a regression-based variation that predicts $g(\theta, \alpha)$ with a linear output layer, but given an α value still chooses the maximizing θ. This method is called NN_r. We also compare to random sampling of θ values, without any training.

GP-LSE is able to learn much more efficiently than the other methods. Figure 28 shows the success rate of the first action parameter vector θ (value 1 if the action
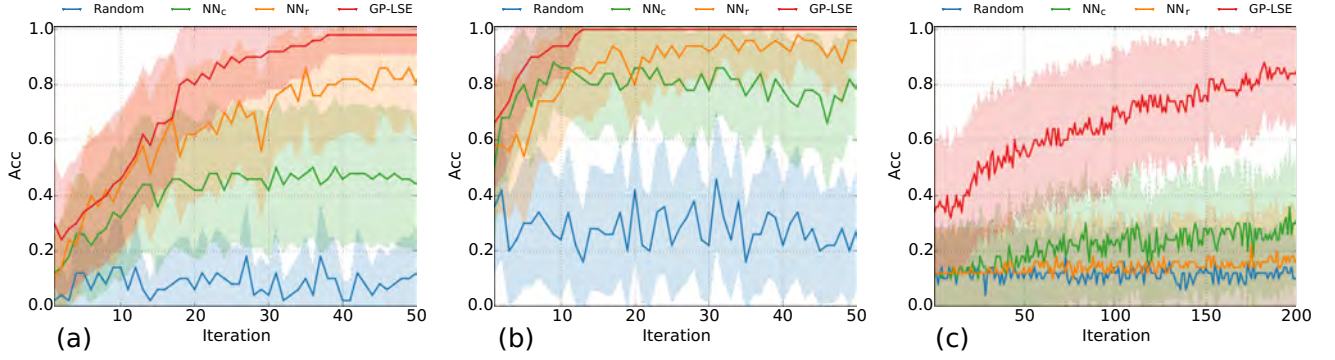
Figure 28. Mean success rate (with 1/2 standard deviation on mean shaded) of the first action recommended by random selection (Random), regression-based neural network (NN_r), classification-based neural network (NN_c) and Gaussian process using level-set estimation (GP-LSE) on (a) a pouring task with 8 parameters (4 are context parameters); (b) a scooping task with 9 parameters (2 are context parameters), and (c) a pushing task with 6 parameters (2 are context parameters).

Figure 29. Comparing the first 5 samples generated by DIVERSE (left) and ADAPTIVE (right) on one of the experiments for pouring. The more transparent the pose, the later it gets sampled.

with parameters θ is actually successful and 0 otherwise) recommended by each of these methods as a function of the number of actively gathered training examples. The results are evaluated through simulation in *Kitchen2D*. GP-LSE recommends its first θ by maximizing the probability that $g(\theta, \alpha) > 0$. The neural-network methods recommend their first θ by maximizing the output value, while RANDOM always selects uniformly randomly from the domain of θ.

In every case, the GP-based method achieves high accuracy well before the others, demonstrating the effectiveness of uncertainty-driven active sampling methods.

C.2.2 Adaptive and diverse sampling: Given a probabilistic estimate of good θ values, obtained by GP-LSE, the next step is to sample values from that set for planning. We compare simple rejection sampling using a uniform proposal distribution (REJECTION), the basic adaptive sampler from section 4.2, and the diversity-aware sampler from section 4.4 with a fixed kernel: the results are shown in Table. 4. The setting for these experiments is exactly as described in section!7.3

In these experiments, as in the ones in section 7.3, DIVERSE uses more samples than ADAPTIVE to achieve 5 positive ones, and its false positive rate is slightly higher than ADAPTIVE, but the diversity of the samples is notably higher. The FP rate of DIVERSE can be decreased by increasing the confidence bound on the level set. We illustrate the ending poses of the 5 pouring actions generated by adaptive sampling with DIVERSE and ADAPTIVE in Figure 29 illustrating that DIVERSE is able to generate more diverse action parameters, which may facilitate planning.

**Table 4.** Effectiveness of adaptive and diverse sampling.

		REJECTION	ADAPTIVE	DIVERSE
Pour	FP (%)	$6.45 \pm 8.06^{*}$	4.04 ± 6.57	5.12 ± 6.94
	T_{50} (s)	$3.10 \pm 1.70^{*}$	0.49 ± 0.10	0.53 ± 0.09
	N_5	$5.51 \pm 1.18^{*}$	5.30 ± 0.92	5.44 ± 0.67
	Diversity	$17.01 \pm 2.90^{*}$	16.24 ± 3.49	18.80 ± 3.38
Scoop	FP (%)	$0.00^{\dagger}$	2.64 ± 6.24	3.52 ± 6.53
	T_{50} (s)	$9.89 \pm 0.88^{\dagger}$	0.74 ± 0.10	0.81 ± 0.11
	N_5	$5.00^{\dagger}$	5.00 ± 0.00	5.10 ± 0.41
	Diversity	$21.1^{\dagger}$	20.89 ± 1.19	21.90 ± 1.04
Push	FP (%)	$68.63 \pm 46.27^{\ddagger}$	21.36 ± 34.18	38.56 ± 37.60
	T_{50} (s)	$7.50 \pm 3.98^{\ddagger}$	3.58 ± 0.99	3.49 ± 0.81
	N_5	$5.00 \pm 0.00^{\ddagger}$	$5.56 \pm 1.51^{\triangle}$	$6.44 \pm 2.11^{\clubsuit}$
	Diversity	$23.06 \pm 0.02^{\ddagger}$	$10.74 \pm 4.92^{\triangle}$	$13.89 \pm 5.39^{\clubsuit}$

*1 out of 50 experiments failed (to generate 50 samples within 10 seconds); †49 out of 50 failed; ‡34 out of 50 failed; 5 out of 16 experiments failed (to generate 5 positive samples within 100 samples); $^{\triangle}$7 out of 50 failed; $^{\clubsuit}$11 out of 50 failed.

Table 5. Effect of distance metric learning on sampling.

Task I	Runtime (ms)	0.2s SR (%)	0.02s SR (%)
ADAPTIVE	8.16 ± 12.16	100.0 ± 0.0	87.1 ± 0.8
DIVERSE-GK	9.63 ± 9.69	100.0 ± 0.0	82.2 ± 1.2
DIVERSE-LK	5.87 ± 4.63	100.0 ± 0.0	99.9 ± 0.1
Task II	Runtime (s)	60s SR (%)	6s SR (%)
ADAPTIVE	3.22 ± 6.51	91.0 ± 2.7	82.4 ± 5.6
DIVERSE-GK	2.06 ± 1.76	95.0 ± 1.8	93.6 ± 2.2
DIVERSE-LK	1.71 ± 1.23	95.0 ± 1.8	94.0 ± 1.5
Task III	Runtime (s)	60s SR (%)	6s SR (%)
ADAPTIVE	5.79 ± 11.04	51.4 ± 3.3	40.9 ± 4.1
DIVERSE-GK	3.90 ± 5.02	56.3 ± 2.0	46.3 ± 2.0
DIVERSE-LK	4.30 ± 6.89	59.1 ± 2.6	49.1 ± 2.6

C.2.3 Learning kernels for diverse sampling: Finally we explore the effectiveness of the diverse sampling algorithm with task-level kernel learning; the setting is analogous to the one in section 7.4. We compare ADAPTIVE, DIVERSE-GK with a fixed kernel, and diverse sampling with learned kernel (DIVERSE-LK), in every case using a high-probability super-level-set estimated by a GP. In DIVERSE-LK, we use $\epsilon = 0.3$. We define the planning reward of a sampler to be $J_k(\phi) = \sum_{n=1}^{\infty} s(\phi, n)\gamma^n$, where $s(\phi, n)$ is the indicator
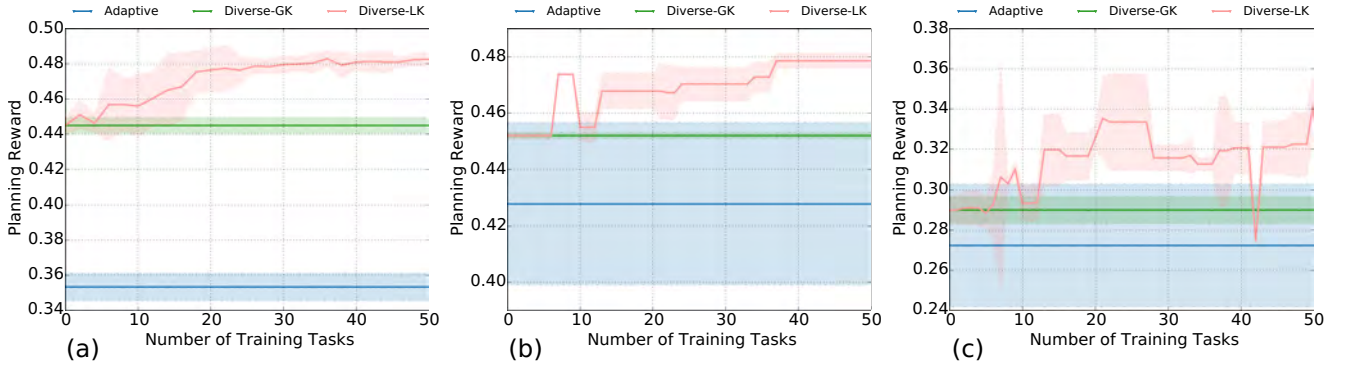
Figure 30. The mean learning curve of reward $J(\phi)$ (with 1.96 standard deviation) as a function of the number of training tasks in three domains: (a) pushing an object off the table (b) pouring into a cup next to a wall (c) picking up a cup in a holder and pour into a cup next to a wall.

variable that the n-th sample from ϕ helped the planner to generate the final plan for a particular task instance k. The reward is discounted by γ^n with $0 < \gamma < 1$, so that earlier samples get higher rewards (we use $\gamma = 0.6$). We average the rewards on tasks drawn from a predefined distribution, and effectively report a lower bound on $J(\phi)$, by setting a time limit on the planner.

The first set of tasks (Task I) we consider is a simple controlled example where the goal is to push an object off a 2D table with the presence of an obstacle on either one side of the table or the other (both possible situations are equally likely). The presence of these obstacles is not represented in the context of the sampler, but the planner will reject sample action instances that generate a collision with an object in the world and request a new sample. We use a fixed range of feasible actions sampled from two rectangles in 2D of unequal sizes. The optimal strategy is to first randomly sample from one side of the table and if no plan is found, sample from the other side.

We show the learning curve of DIVERSE-LK with respect to the planning reward metric $J(\phi)$ in figure 30 (a). 1000 initial arrangements of obstacles were drawn randomly for testing. We also repeat the experiments 5 times to obtain the 95% confidence interval. For DIVERSE-GK, the kernel inverse is initialized as $[1, 1]$ and if, for example, it sampled on the left side of the object (pushing to the right) and the obstacle is on the right, it may not choose to sample on the right side because the kernel indicates that the other feature is has more diversity. However, after a few planning instances, DIVERSE-LK is able to figure out the right configuration of the kernel and its sampling strategy becomes the optimal one.

We also tested these three sampling algorithms on two more complicated tasks. We select a fixed test set with 50 task specifications and repeat the evaluation 5 times. The first one (Task II) involves picking up cup A, getting water from a faucet, move to a pouring position, pour water into cup B, and finally placing cup A back in its initial position. Cup B is placed randomly either next to the wall on the left or right. The second task is a harder version of Task II, with the additional constraint that cup A has a holder and the sampler also has to figure out that the grasp location must be close to the top of the cup (Task III).

We show the learning results in figure 30 (b) and (c) and timing results in table 5 (after training). We conjecture that the sharp turning points in the learning curves of Tasks II

and III are a result of high penalty on the kernel length scales and the limited size (50) of the test tasks, and we plan to investigate more in the future work. Nevertheless, DIVERSE-LK is still able to find a better solution than the alternatives in Tasks II and III. Moreover, the two diverse sampling methods achieve lower variance on the success rate and perform more stably after training.

C.2.4 Integration Finally, we integrate the learned action sampling models for pour and scoop with 7 pre-existing robot operations (move, push, pick, place, fill, dump, stir) in a domain specification for PDDLStream. The robot's goal is to "serve" a cup of coffee with cream and sugar by placing it on the green coaster near the edge of the table. Accomplishing this requires general-purpose planning, including picking where to grasp the objects, where to place them back down on the table, and what the pre-operation poses of the cups and spoon should be before initiating the sensorimotor primitives for pouring and scooping should be. Significant perturbations of the object arrangements are handled without difficulty. For example, We use the focused algorithm within PDDLStream, and it solves the task in 20-40 seconds for a range of different arrangements of objects. Some resulting plans and execution sequences can be found at `https://ziw.mit.edu/projects/kitchen2d/`.

In summary, our experiments in *Kitchen2D* illustrate a critical ability: to augment the existing competences of a robotic system (such as picking and placing objects) with new sensorimotor primitives by learning probabilistic models of their preconditions and effects and using a state-of-the-art domain-independent continuous-space planning algorithm to combine them fluidly and effectively to achieve complex goals.

References

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M and others (2016) Tensorflow: A system for large-scale machine learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. pp. 265–283.

Affandi RH, Fox E, Adams R and Taskar B (2014) Learning the parameters of determinantal point process kernels. In: *International Conference on Machine Learning (ICML)*.

Bogunovic I, Scarlett J, Krause A and Cevher V (2016) Truncated variance reduction: A unified approach to Bayesian optimization and level-set estimation. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Brandi S, Kroemer O and Peters J (2014) Generalizing pouring actions between objects using warped parameters. In: *Humanoids*.

Bryan B, Nichol RC, Genovese CR, Schneider J, Miller CJ and Wasserman L (2006) Active learning for identifying function threshold boundaries. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Calli B, Singh A, Walsman A, Srinivasa S, Abbeel P and Dollar AM (2015) The YCB object and model set: Towards common benchmarks for manipulation research. In: *IEEE International Conference on Advanced Robotics (ICAR)*.

Catto E (2011) Box2D, A 2D Physics Engine for Games. `http://box2d.org`.

Chitnis R, Hadfield-Menell D, Gupta A, Srivastava S, Groshev E, Lin C and Abbeel P (2016) Guided Search for Task and Motion Plans Using Learned Heuristics. *IEEE International Conference on Robotics and Automation (ICRA)* .

Coumans E and Bai Y (2016–2019) Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`.

Diankov R (2010) *Automated construction of robotic manipulation programs*. PhD Thesis, Robotics Institute, Carnegie Mellon University.

Diankov R and Kuffner J (2008) OpenRAVE: A Planning Architecture for Autonomous Robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University.

Englert P and Toussaint M (2016) Combined optimization and reinforcement learning for manipulation skills. In: *Robotics: Science and Systems Conference (RSS)*.

Fischler MA and Bolles RC (1981) Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM* 24(6): 381–395.

Foster DP and Vohra R (1999) Regret in the on-line decision problem. *Games and Economic Behavior* 29(1-2).

Gal Y and Ghahramani Z (2016) Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In: *International Conference on Machine Learning (ICML)*.

Garrett CR, Chitnis R, Holladay R, Kim B, Silver T, Kaelbling LP and Lozano-Pérez T (2021) Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems* 4.

Garrett CR, Lozano-Pérez T and Kaelbling LP (2017a) Sample-Based Methods for Factored Task and Motion Planning. In: *Robotics: Science and Systems Conference (RSS)*.

Garrett CR, Lozano-Pérez T and Kaelbling LP (2020a) PDDL-Stream: Integrating Symbolic Planners and Blackbox Samplers. In: *International Conference on Automated Planning and Scheduling (ICAPS)*.

Garrett CR, Lozano-Pérez T and Kaelbling LPL (2017b) Sampling-based methods for factored task and motion planning. In: *The International Journal of Robotics Research*.

Garrett CR, Paxton C, Lozano-Pérez T, Kaelbling LP and Fox D (2020b) Online Replanning in Belief Space for Partially Observable Task and Motion Problems. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Glover J (2014) *The quaternion Bingham distribution, 3D object detection, and dynamic manipulation*. PhD Thesis, Massachusetts Institute of Technology.

Gotovos A, Casati N, Hitz G and Krause A (2013) Active learning for level set estimation. In: *International Conference on Artificial Intelligence (IJCAI)*.

GPy (2012) GPy: A Gaussian process framework in python. `http://github.com/SheffieldML/GPy`.

Hafiz Affandi R, Fox EB and Taskar B (2013) Approximate Inference in Continuous Determinantal Point Processes. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Hermans T, Li F, Rehg JM and Bobick AF (2013) Learning contact locations for pushing and orienting unknown objects. In: *Humanoids*.

Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S and others (2017) Speed/accuracy trade-offs for modern convolutional object detectors. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kaelbling LP and Lozano-Pérez T (2011) Hierarchical task and motion planning in the now. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Kaelbling LP and Lozano-Pérez T (2013) Integrated task and motion planning in belief space. *International Journal of Robotics Research (IJRR)* .

Kaelbling LP and Lozano-Perez T (2017) Learning composable models of parameterized skills. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Kapoor A, Grauman K, Urtasun R and Darrell T (2007) Active learning with gaussian processes for object categorization. In: *International Conference on Computer Vision (ICCV)*. IEEE.

Kim B, Kaelbling LP and Lozano-Perez T (2017) Learning to guide task and motion planning using score-space representation. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Kim B, Kaelbling LP and Lozano-Perez T (2018) Guiding Search in Continuous State-action Spaces by Learning an Action Sampler from Off-target Search Experience. In: *AAAI Conference on Artificial Intelligence*.

Konidaris G, Kaelbling LP and Lozano-Perez T (2018) From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal or Artificial Intelligence Research* 61.

Kroemer O and Sukhatme G (2016a) Meta-level priors for learning manipulation skills with sparse features. In: *International Symposium on Experimental Robotics (ISER)*.

Kroemer O and Sukhatme GS (2016b) Learning spatial preconditions of manipulation skills using random forests. In: *Humanoids*.

Kuffner Jr JJ and LaValle SM (2000) {RRT-Connect}: An efficient approach to single-query path planning. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Kulesza A, Taskar B and others (2012) Determinantal point processes for machine learning. *Foundations and Trends in*

Machine Learning 5(2–3).

Lakshminarayanan B, Pritzel A and Blundell C (2017) Simple and scalable predictive uncertainty estimation using deep ensembles. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollár P and Zitnick CL (2014) Microsoft COCO: Common objects in context. In: *Europ. Conference on Computer Vision (ECCV)*.

McDermott D, Ghallab M, Howe A, Knoblock C, Ram A, Veloso M, Weld D and Wilkins D (1998) PDDL: The Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control.

Neal RM (2012) *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

O'Callaghan ST and Ramos FT (2012) Gaussian process occupancy maps. *The International Journal of Robotics Research* .

OpenAI, Akkaya I, Andrychowicz M, Chociej M, Litwin M, McGrew B, Petron A, Paino A, Plappert M, Powell G, Ribas R, Schneider J, Tezak N, Tworek J, Welinder P, Weng L, Yuan Q, Zaremba W and Zhang L (2019) Solving Rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113* .

Pan Z, Park C and Manocha D (2016) Robot Motion Planning for Pouring Liquids. In: *International Conference on Automated Planning and Scheduling (ICAPS)*.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E and others (2011) Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12(Oct): 2825–2830.

Quigley M, Gerkey B, Conley K, Faust J, Foote T, Leibs J, Berger E, Wheeler R and Ng AY (2009) {ROS}: an open-source Robot Operating System. In: *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open-Source Software*.

Rasmussen CE and Williams CKI (2006) Gaussian processes for machine learning. *The MIT Press* .

Ren S, He K, Girshick R and Sun J (2015) Faster r-cnn: Towards real-time object detection with region proposal networks. In: *Advances in Neural Information Processing Systems (NeurIPS)*. pp. 91–99.

Rusu RB and Cousins S (2011) 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Schenck C and Fox D (2017) Visual closed-loop control for pouring liquids. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Schenck C, Tompson J, Fox D and Levine S (2017) Learning Robotic Manipulation of Granular Media. In: *Conference on Robot Learning (CoRL)*.

Schreiter J, Nguyen-Tuong D, Eberts M, Bischoff B, Markert H and Toussaint M (2015) Safe exploration for active learning with gaussian processes. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

Srinivas N, Krause A, Kakade SM and Seeger M (2010) Gaussian process optimization in the bandit setting: No regret and experimental design. In: *International Conference on Machine Learning (ICML)*.

Srivastava S, Fang E, Riano L, Chitnis R, Russell S and Abbeel P (2014) Combined Task and Motion Planning Through an Extensible Planner-Independent Interface Layer. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Stilman M (2010) Global manipulation planning in robot joint space with task constraints. *IEEE Transactions on Robotics* 26(3): 576–584. DOI:10.1109/TRO.2010.2044949.

Sutton RS, Precup D and Singh S (1999) Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1-2): 181–211.

Tamosiunaite M, Nemec B, Ude A and Wörgötter F (2011) Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robotics and Autonomous Systems* 59(11).

Vasudevan S, Ramos F, Nettleton E, Durrant-Whyte H and Blair A (2009) Gaussian process modeling of large scale terrain. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Wang AS and Kroemer O (2019) Learning robust manipulation strategies with multimodal state transition models and recovery heuristics. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Wang Z, Garrett C, Kaelbling L and Lozano-Perez T (2018) Active Model Learning and Diverse Action Sampling for Task and Motion Planning. In: *The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Wang Z, Zhou B and Jegelka S (2016) Optimization as estimation with Gaussian processes in Bandit settings. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Wipf DP and Nagarajan SS (2008) A new view of automatic relevance determination. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Xia V, Wang Z, Allen K, Silver T and Kaelbling LP (2019) Learning sparse relational transition models. In: *International Conference on Learning Representations (ICLR)*.

Yamaguchi A and Atkeson CG (2016) Differential dynamic programming for graph-structured dynamical systems: Generalization of pouring behavior with different skills. In: *Humanoids*.

Yamaguchi A, Atkeson CG, Niekum S and Ogasawara T (2014) Learning pouring skills from demonstration and practice. In: *IEEE-RAS International Conference on Humanoid Robots*.

Yao Z and Gupta K (2005) Path planning with general end-effector constraints: Using task space to guide configuration space search. In: *The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.