Active model learning and diverse action sampling for task and motion planning

Zi Wang Caelan Reed Garrett Leslie Pack Kaelbling Tomás Lozano-Pérez

***Abstract*— The objective of this work is to augment the basic abilities of a robot by learning to use new sensorimotor primitives to enable the solution of complex long-horizon problems. Solving long-horizon problems in complex domains requires flexible generative planning that can combine primitive abilities in novel combinations to solve problems as they arise in the world. In order to plan to combine primitive actions, we must have models of the preconditions and effects of those actions: under what circumstances will executing this primitive 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 conditions of operator effectiveness from small numbers of expensive training examples collected by experimentation on a robot. We develop adaptive sampling methods for generating diverse elements of continuous sets (such as robot configurations and object poses) during planning for solving a new task, so that planning is as efficient as possible. We demonstrate these methods in an integrated system, combining newly learned models with an efficient continuous-space robot task and motion planner to learn to solve long horizon problems more efficiently than was previously possible.

I. Introduction

For a robot to be effective in a domain that combines novel sensorimotor primitives, such as pouring or stirring, with long-horizon, high-level task objectives, such as cooking a meal or making a cup of coffee, it is necessary to acquire models of these primitives to use in planning robot motions and manipulations. These models characterize (a) conditions under which the primitive is likely to succeed and (b) the effects of the primitive on the state of the world.

Figure 1 illustrates several instances of a parameterized motor primitive for pouring in a simple two-dimensional domain. The primitive action 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 $(c_R, p_A, s_A, p_B, s_B, \theta, c'_R, p'_A) \in \chi$ is the action feasible: one key objective of our work is to efficiently learn a representation of the feasible region χ.

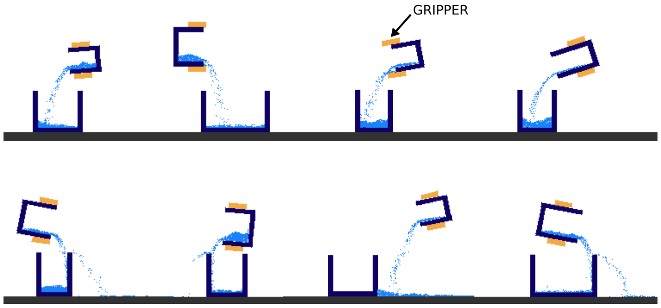

Fig. 1: Several examples of executing a pouring primitive with different settings, including control parameters, cup sizes, and relative placements.

For learning this model, each training example requires running the primitive, which is expensive on real robot hardware and even in high-fidelity simulation. To minimize the amount of training data required, we actively select each setting in which the primitive is executed, with the goal of obtaining as much information as possible about how to use the primitive. This results in a dramatic reduction in required examples over our preliminary work [1] on this problem.

Given a model of a primitive, embodied in χ, we utilize existing sample-based algorithms for *task and motion planning* (TAMP) to find plans. To use the model within the planner, it is necessary to select candidate instances of the primitives for expansion during the search for a plan. The objective here is not to gain information, but to select primitive instances that are likely to be successful. It is not enough to select a single instance, however, because there may be other circumstances that make a particular instance infeasible within a planning context: for example, although the most reliable way to grasp a particular object might be from the top, the robot might encounter the object situated in a cupboard in a way that makes the top grasp infeasible. Thus, our action-sampling mechanism must select instances that are both likely to succeed and are *diverse* from one another, so that the planner has coverage of the space of possible actions.

One difficulty in sampling χ is that it inhabits a lower-dimensional submanifold of the space it is defined in, because some relations among robot configurations and object poses, for example, are functional. The STRIPStream planner [2], [3] introduced a strategy for sampling from such *dimensionality-reducing* constraints by constructing *conditional samplers* that, given values of some variables, generate values of the other variables that satisfy the constraint. Our goal in this paper is to learn and use conditional samplers within the STRIPStream planner.

*Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA 02139. {ziw,caelan,lpk,tlp}@csail.mit.edu.

Our technical strategy for addressing the problems of (a) learning success constraints and (b) generating diverse samples is based on an explicit representation of uncertainty about an underlying *scoring* function that measures the quality or likelihood of success of a parameter vector, and uses Gaussian process (GP) techniques to sample for information-gathering during learning and for success probability and diversity during planning. We begin by describing some basic background, discuss related work, describe our methods in technical detail, and then present experimental results of learning and planning with several motor primitives in a two-dimensional dynamic simulator.

II. Problem formulation and background

We will focus on the formal problem of learning and using a conditional sampler of the form $\phi(\theta \mid \alpha)$, where $\alpha \in R^{d_\alpha}$ is a vector of contextual parameters and $\theta \in B$ is a vector of parameters that are to be generated, conditioned on α. We assume in the following that the domain of θ is a hyper-rectangular space $B = [0,1]^{d_\theta} \subset R^{d_\theta}$, but generalization to other topologies is possible. The conditional sampler generates samples of θ such that $(\theta, \alpha) \in \chi$ where $\chi \subset R^{d_\alpha + d_\theta}$ characterizes the set of world states and parameters for which the skill is feasible. We assume that χ can be expressed in the form of an inequality constraint $\chi(\theta, \alpha) = (g(\theta, \alpha) > 0)$, where $g(\cdot)$ is a scoring function with arguments θ and α. We denote the *super level-set* of the scoring function given α by $A_\alpha = \{\theta \in B \mid g(\theta, \alpha) > 0\}$. For example, the scoring function $g(\cdot)$ for pouring might be the proportion of poured liquid that actually ends up in the target cup, minus some target proportion. We assume the availability of values of such a score function during training rather than just binary labels of success or failure. In the following, we give basic background on two important components of our method: Gaussian processes and STRIPStream.

Gaussian processes (GPs) are distributions over functions, and popular priors for Bayesian non-parametric regression. In a GP, any finite set of function values has a multivariate Gaussian distribution. In this paper, we use the Gaussian process $\mathrm{GP}(0,k)$ which has mean zero and covariance (kernel) function $k(\boldsymbol{x}, \boldsymbol{x}')$. Let f be a true underlying function sampled from $\mathrm{GP}(0,k)$. 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 $\sigma^2(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}^{\mathcal{D}}(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{K}^{\mathcal{D}} + \zeta^2 \boldsymbol{I})^{-1}\boldsymbol{k}_t(\boldsymbol{x}')$ 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 $\boldsymbol{k}^{\mathcal{D}}(\boldsymbol{x}) = [k(\boldsymbol{x}_i, \boldsymbol{x})]_{\boldsymbol{x}_i \in \mathcal{D}}$ [4]. 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 $\mathrm{GP}(\mu, \sigma)$.

STRIPStream [3] is a framework for incorporating black-box sampling procedures in a planning language. It extends the STRIPS planning language [5] by adding *streams*, declarative specifications of conditional generators. Streams have previously been used to model off-the-shelf motion planners, collision checkers, inverse kinematic solvers. In this work, we learn new conditional generators, such as samplers for pouring, and incorporate them using streams.

III. Related Work

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

There is a large amount of work on learning individual motor primitives such pushing [6], [7], scooping [8], and pouring [9], [10], [11], [12], [13]. 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 [1], where constraint-based pre-image models are learned for parameterized action primitives, by giving a probabilistic characterization of the pre-image and using these models during planning.

Other approaches exist to learning models of the preconditions and effects of sensorimotor skills suitable for planning. One [14] constructs a completely symbolic model of skills that enable purely symbolic task planning. Our method, on the other hand, learns hybrid models, involving continuous parameters. Another [15] learns image classifiers for pre-conditions but does not support general-purpose planning.

We use GP-based level set estimation [16], [17], [4], [18] to model the feasible regions (super level set of the scoring function) of action parameters. We use the *straddle* algorithm [16] to actively sample from the function threshold, in order to estimate the super level set that satisfy the constraint with high probability. Our methods can be extended to other function approximators that gives uncertainty estimates, such as Bayesian neural networks and their variants [19], [20].

Determinantal point processes (DPPs) [21] are typically used for diversity-aware sampling. However, both sampling from a continuous DPP [22] and learning the kernel of a DPP [23] are challenging.

Several approaches to TAMP utilize generators to enumerate infinite sequences of values [24], [25], [2]. Our learned samplers can be incorporated in any of these approaches. Additionally, some recent papers have investigated learning effective samplers within the context of TAMP. Chitnis et al. [26] 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. [27] proposed a method for selecting from a discrete set of samples by scoring new samples based on their correlation with previously attempted samples. In subsequent work, they instead train a Generative Adversarial Network to directly produce a distribution of satisfactory samples [28].

IV. Active sampling for learning and planning

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 GP on the score function g to select informative queries using a level-set estimation approach. Our objective in the planning phase is to select a diverse set of samples $\{\theta_i\}$ for which it is likely that $\theta \in A_\alpha$. We do this in two steps: first, we use a novel risk-aware sampler to generate θ values that satisfy the constraint with high probability; second, we integrate this sampler with STRIPStream, where we generate samples from this set that represent its diversity, in order to expose the full variety of choices to the planner.

A. Actively learning the constraint with a GP

Our goal is to be able to sample from the super level set $A_\alpha = \{\theta \in B \mid g(\theta, \alpha) > 0\}$ for any given context α, which requires learning the decision boundary $g(\theta, \alpha) = 0$. During training, we select α values from a distribution reflecting naturally occurring contexts in the underlying domain. Note that learning the level-set is a different objective from learning all of the function values well, 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 [16] to actively select samples of θ for evaluation by running the motor primitive. 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. The straddle algorithm selects θ that maximizes the acquisition function $\psi(\theta; \alpha, \mu, \sigma) = -|\mu(\theta, \alpha)| + 1.96\sigma(\theta, \alpha)$. It has a high value for values of θ that are near the boundary for the given α or for which the score function is highly uncertain. The parameter 1.96 is selected such that if $\psi(\theta)$ 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 [18], [17]. After each new evaluation, we retrain the Gaussian process by maximizing its marginal data-likelihood with respect to its hyper-parameters. Alg. 1 specifies the algorithm; GP-predict$(\cdot)$ computes the posterior mean and variance as explained in Sec. II.

Algorithm 1 Active Bayesian Level Set Estimation

```
Given initial data set $\mathcal{D}$, context $\alpha$, desired number of samples $T$
for $t = 1 \to T$ do
    $\mu, \sigma \leftarrow$ GP-predict($\mathcal{D}$)
    $\theta \leftarrow \arg\max_\theta \psi(\theta; \alpha, \mu, \sigma)$
    $y \leftarrow g(\theta, \alpha)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{((\theta, \alpha), y)\}$
return $\mathcal{D}$
```

B. Risk-aware adaptive sampling for constraint satisfaction

Now we can use this Bayesian estimate of the scoring function g to select action instances for planning. Given a new context α, which need not have occured 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, $g(\theta, \alpha) \geq 0$. 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. The following is a direct corollary of Lemma 3.2 of [29].

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

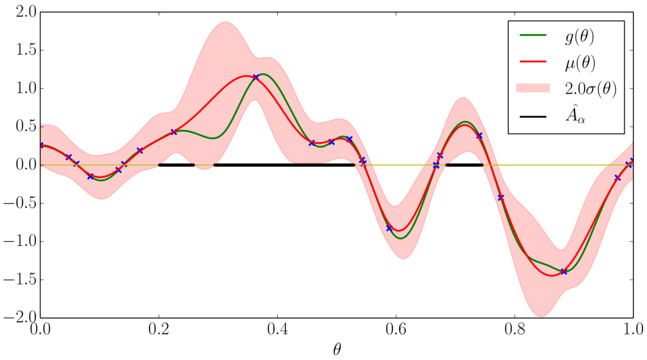

Fig. 2: High-probability super-level-set in black.

Define the high-probability super-level-set of θ given context α as $\hat{A}_\alpha = \{\theta \mid \mu(\theta, \alpha) > \beta^*\,\sigma(\theta, \alpha)\}$ 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 any given α, using the definition of β^* from Corollary 1, the set $\hat{A}_\alpha$ may be empty. In that case, we can relax our criterion to include the set of θ values whose score is within 5% of the θ value that is currently estimated to have the highest likelihood of satisfying the constraint: $\beta = \Phi^{-1}(0.95\Phi(\max_{\theta \in B} \mu(\theta, \alpha)/\sigma(\theta, \alpha)))$ where Φ is the cumulative density function of a normal distribution.

Figure 2 illustrates the computation of $\hat{A}_\alpha$. The green line is the true hidden $g(\theta)$; the blue × 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.

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 typically 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 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 and 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 generates a sample each time it is called. The main work is done by SAMPLEBUFFER, 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 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 2 Super Level Set Adaptive Sampling

```
function ADAPTIVESAMPLER($\mu, \sigma, \alpha$)
    $\beta \leftarrow \Phi^{-1}(0.95\Phi(\max_{\theta \in B} \mu(\theta, \alpha)/\sigma(\theta, \alpha)))$
    $\Theta_{init} \leftarrow \{\arg\max_{\theta \in B} \frac{\mu(\theta)}{\sigma(\theta)}\}$; $\Theta \leftarrow \emptyset$
    while True do
        if $|\Theta| < m/2$ then
            $\Theta \leftarrow$ SAMPLEBUFFER($\mu, \sigma, \alpha, \beta, \Theta_{init}, n, m$)
        $\theta \leftarrow \Theta[0]$
        yield $\theta$
        $\Theta \leftarrow \Theta \setminus \{\theta\}$
function SAMPLEBUFFER($\mu, \sigma, \alpha, \beta, \Theta_{init}$)
    $v \leftarrow [1]_{d=1}^{d_\theta}$; $\Theta \leftarrow \Theta_{init}$; $p \leftarrow [1]_{i=1}^{|\Theta|}$
    while True do
        $\Theta' \leftarrow$ SampleTGMM($n; p, \Theta, v, B$)
        $\Theta_a \leftarrow \{\theta \in \Theta' \mid \mu(\theta) > \beta\sigma(\theta)\}$
        $p_a \leftarrow 1/p_{\text{TGMM}}(\Theta_a; p, \Theta, v, B)$
        $v \leftarrow v/2$ if $|\Theta_a| < |\Theta'|/2$ else $v \times 2$
        $\Theta'' \leftarrow$ SampleUniform($n; B$)
        $\Theta_r \leftarrow \{\theta \in \Theta'' \mid \mu(\theta) > \beta\sigma(\theta)\}$
        $p_r \leftarrow [Vol(B)]_{i=1}^{|\Theta_r|}$
        $p \leftarrow$ Normalize($[p, p_r, p_a]$)
        $\Theta \leftarrow [\Theta, \Theta_r, \Theta_a]$
        if $|\Theta| > m$ then
            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, n is finite, but this method is much more efficient than rejection sampling.

C. 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 for continuous action spaces. Such planners perform backtracking search, potentially needing to consider multiple different parameterized instances of a particular action before finding one that will work well in the overall context of the planning problem. The efficiency of this process depends on the order in which samples of action instances are generated. Intuitively, when previous samples of this action for this context 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 from those that were previously tried as possible. 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) [21]. We begin with similar ideas and adapt them to the planning domain, quantifying 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 $\Xi^S_{ij} = \xi(\theta_i, \theta_j), \forall \theta_i, \theta_j \in S$, 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 $\xi(\theta_i, \theta_j)\zeta^{-2} + \mathbf{1}_{\theta_i \equiv \theta_j}$ 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 [30]. 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 $D([\theta_i]_{i=1}^N) \geq (1 - \frac{1}{e}) \max_{|S| \leq N} D(S)$ $\forall N = 1, 2, \cdots$, 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, to return, 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 Alg 4, to yield a kernel η.

Algorithm 3 Super Level Set Diverse Sampling

```
function DIVERSESAMPLER($\mu, \sigma, \alpha, \eta$)
    $\beta \leftarrow \lambda(\max_{\theta \in B} \frac{\mu(\theta)}{\sigma(\theta)})$; $\Theta \leftarrow \emptyset$
    $\theta \leftarrow \arg\max_{\theta \in B} \frac{\mu(\theta)}{\sigma(\theta)}$; $S \leftarrow \emptyset$
    while planner requires samples do
        yield $\theta$, S
        if $|\Theta| < m/2$ then
            $\Theta \leftarrow$ SAMPLEBUFFER($\mu, \sigma, \alpha, \beta, \Theta_{init}$)
        $S \leftarrow S \cup \{\theta\}$                    ▷ $S$ contains samples before $\theta$
        $\theta \leftarrow \arg\max_{\theta \in \Theta} \eta_S(\theta)$
        $\Theta \leftarrow \Theta \setminus \{\theta\}$
```

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 which samples are accepted by the planner and which are not. We derive our notion of similarity by assuming that all samples that are rejected by the planner 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 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 lengthscale. 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 [31] 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 lengthscale 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 4 Task-level Kernel Learning

```
for task in T do
    $\alpha \leftarrow$ current context
    $\mu, \sigma \leftarrow$ GP-predict($\alpha$); $S \leftarrow \emptyset$
    while plan not found do
        if $|S| > 0$ then
            $d \leftarrow \arg\max_{d \in [d_\theta]} \tau_S^\theta(d)$
            $l_d \leftarrow (1-\epsilon)l_d$
        $\theta, S \leftarrow$ DiverseSampler($\mu, \sigma, \alpha, \xi(\cdot, \cdot; l)$)
        Check if a plan exist using $\theta$
```

Alg. 4 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 the next section.

V. Experiments

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.

To test our algorithms, we implemented a simulated 2D kitchen based on the physics engine Box2D [32]. Fig. 3 shows several scenes indicating the variability of arrangements of objects in the domain. We use bi-directional RRT [33] to implement motion planning. The parameterized primitive motor 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 degrees of freedom (2D position and rotation). The material to be poured or scooped is simulated as small circular particles.

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 actions are represented by a trajectory of way points for the gripper, relative to the object it is interacting with. For pouring, we use the scoring function $g_{pour}(x) = \exp(2 * (x * 10 - 9.5)) - 1$, 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 Sec. V-A has the same setting as [1], viewing the gripper/object with a bird-eye view. We will make the code for the simulation and learning methods public at `https://github.com/zi-w/Kitchen2D`.

A. Active learning for conditional samplers

We 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. For comparison, we also implemented a simple method [1], 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 $\theta^* = \arg\max_\theta \mathrm{NN}(\alpha, \theta)$ 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. Fig. 4 shows the accuracy of the first action

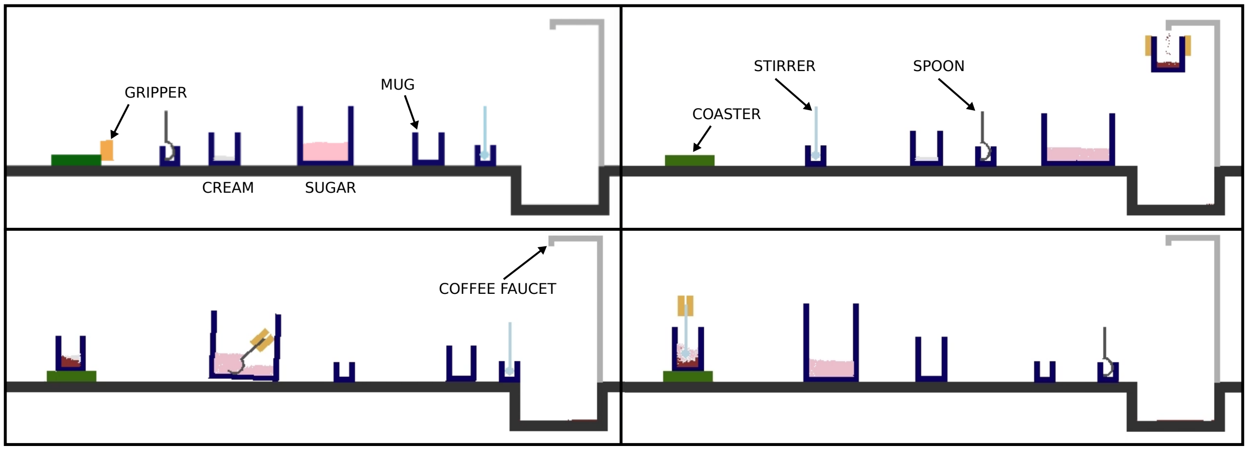

Fig. 3: Four arrangements of objects in 2D kitchen, including: green coaster, coffee faucet, yellow robot grippers, sugar scoop, stirrer, coffee mug, small cup with cream, larger container with pink sugar.

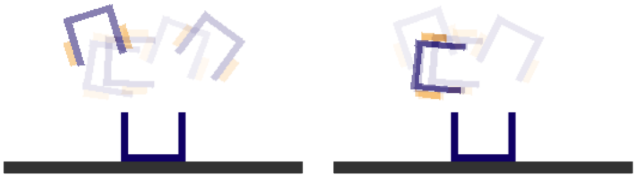

Fig. 5: Comparing the first 5 samples generated by DIVERSE (left) and ADAPTIVE (right) on one of the experiments for pouring. The more transparent the pose, the later it gets sampled.

TABLE I: Effectiveness of adaptive and diverse sampling.

		REJECTION	ADAPTIVE	DIVERSE
Pour	FP (%)	6.45 ± 8.06*	4.04 ± 6.57	5.12 ± 6.94
	T_{50} (s)	3.10 ± 1.70*	0.49 ± 0.10	0.53 ± 0.09
	N_5	5.51 ± 1.18*	5.30 ± 0.92	5.44 ± 0.67
	Diversity	17.01 ± 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.

parameter vector θ (value 1 if the action 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. 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 perfect or high accuracy well before the others, demonstrating the effectiveness of uncertainty-driven active sampling methods.

B. Adaptive sampling 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 Sec. IV-B, and the diversity-aware sampler from Sec. IV-C with a fixed kernel: the results are shown in Table. I.

We report the false positive rate (proportion of samples that do not satisfy the true constraint) on 50 samples (FP), 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. We limit cpu time for gathering 50 samples to 10 seconds (running with Python 2.7.13 and Ubuntu 14.04 on Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz with 64GB memory.) If no sample is returned within 10 seconds, we do not include that experiment in the reported results except the sampling time. Hence the reported sampling time may be a lower bound on the actual sampling time. The diversity term is measured by $D(S) = \log\det(\Xi^S\zeta^{-2} + \boldsymbol{I})$ using a squared exponential kernel with inverse lengthscale $l = [1, 1, \cdots, 1]$ 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 $D(S)$. We also report the total number of samples needed to achieve 5 positive ones (N_p). If the method is not able to get 5 positive samples within 100 samples, we report failure and do not include them in the diversity metric or the N_p metric.

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. We

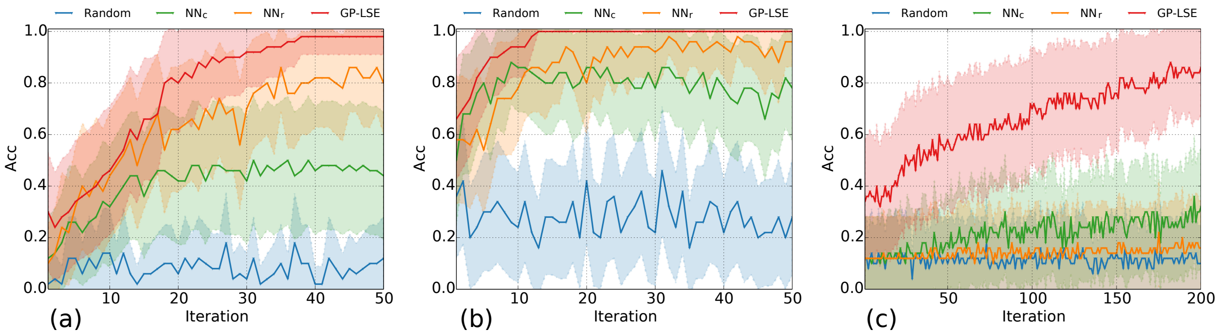

Fig. 4: Mean accuracy (with 1/2 stdev on mean shaded) of the first action recommended by each method.

TABLE II: 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

illustrate the ending poses of the 5 pouring actions generated by adaptive sampling with DIVERSE and ADAPTIVE in Fig. 5illustrating that DIVERSE is able to generate more diverse action parameters, which may facilitate planning.

C. Learning kernels for diverse sampling in planning

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 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 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 Fig. 6 (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 Fig. 6 (b) and (c) and timing results in Tab. II (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 lengthscales 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.

D. Integrated system

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 STRIPStream. The robot's goal is to "serve" a cup of coffee with cream and sugar by placing it on the

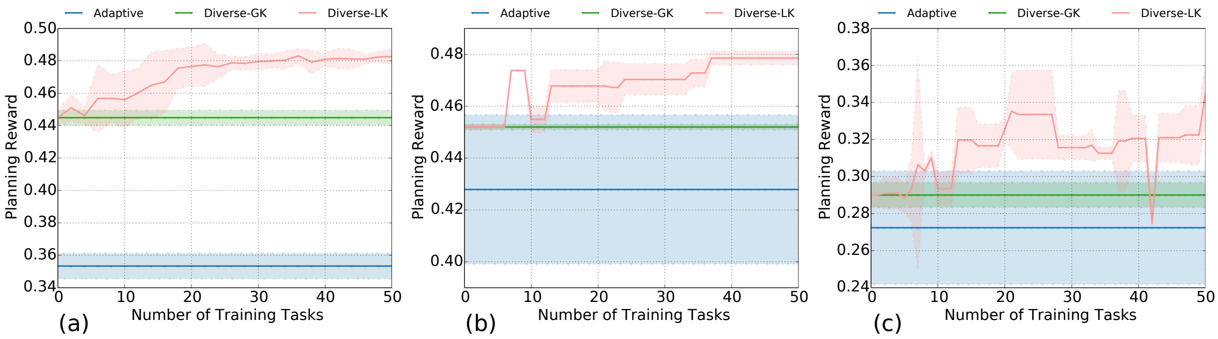

Fig. 6: The mean learning curve of reward $J(\phi)$ (with 1.96 standard deviation) as a function of the number of training tasks in three domains: (a) pushing an object off the table (b) pouring into a cup next to a wall (c) picking up a cup in a holder and pour into a cup next to a wall.

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[1]. A resulting plan and execution sequence is shown in the accompanying video.

This work illustrates 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

[1] L. P. Kaelbling and T. Lozano-Perez, "Learning composable models of parameterized skills," in *ICRA*, 2017.

[2] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "Sample-based methods for factored task and motion planning," in *RSS*, 2017.

[3] ——, "Strips planning in infinite domains," *arXiv:1701.00287*, 2017.

[4] C. E. Rasmussen and C. K. Williams, "Gaussian processes for machine learning," *The MIT Press*, 2006.

[5] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, pp. 189–208, 1971.

[6] O. Kroemer and G. Sukhatme, "Meta-level priors for learning manipulation skills with sparse features," in *ISER*, 2016.

[7] T. Hermans, F. Li, J. M. Rehg, and A. F. Bobick, "Learning contact locations for pushing and orienting unknown objects," in *Humanoids*, 2013.

[8] C. Schenck, J. Tompson, D. Fox, and S. Levine, "Learning robotic manipulation of granular media," in *CORL*, 2017.

[9] Z. Pan, C. Park, and D. Manocha, "Robot motion planning for pouring liquids." in *ICAPS*, 2016.

[10] M. Tamosiunaite, B. Nemec, A. Ude, and F. Wörgötter, "Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives," *Robotics and Autonomous Systems*, vol. 59, no. 11, 2011.

[11] S. Brandi, O. Kroemer, and J. Peters, "Generalizing pouring actions between objects using warped parameters," in *Humanoids*, 2014.

[12] A. Yamaguchi and C. G. Atkeson, "Differential dynamic programming for graph-structured dynamical systems: Generalization of pouring behavior with different skills," in *Humanoids*, 2016.

[13] C. Schenck and D. Fox, "Visual closed-loop control for pouring liquids," in *ICRA*, 2017.

[14] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *JAIR*, vol. 61, 2018.

[15] O. Kroemer and G. S. Sukhatme, "Learning spatial preconditions of manipulation skills using random forests," in *Humanoids*, 2016.

[16] B. Bryan, R. C. Nichol, C. R. Genovese, J. Schneider, C. J. Miller, and L. Wasserman, "Active learning for identifying function threshold boundaries," in *NIPS*, 2006.

[17] A. Gotovos, N. Casati, G. Hitz, and A. Krause, "Active learning for level set estimation," in *IJCAI*, 2013.

[18] I. Bogunovic, J. Scarlett, A. Krause, and V. Cevher, "Truncated variance reduction: A unified approach to bayesian optimization and level-set estimation," in *NIPS*, 2016.

[19] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *ICML*, 2016.

[20] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *NIPS*, 2017.

[21] A. Kulesza, B. Taskar, *et al.*, "Determinantal point processes for machine learning," *Foundations and Trends in Machine Learning*, vol. 5, no. 2–3, 2012.

[22] R. Hafiz Affandi, E. B. Fox, and B. Taskar, "Approximate inference in continuous determinantal point processes," in *NIPS*, 2013.

[23] R. H. Affandi, E. Fox, R. Adams, and B. Taskar, "Learning the parameters of determinantal point process kernels," in *ICML*, 2014.

[24] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, 2011.

[25] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *ICRA*, 2014.

[26] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," in *ICRA*, 2016.

[27] B. Kim, L. P. Kaelbling, and T. Lozano-Perez, "Learning to guide task and motion planning using score-space representation," in *ICRA*, 2017.

[28] ——, "Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience," in *AAAI*, 2018.

[29] Z. Wang, B. Zhou, and S. Jegelka, "Optimization as estimation with Gaussian processes in bandit settings," in *AISTATS*, 2016.

[30] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *ICML*, 2010.

[31] D. P. Foster and R. Vohra, "Regret in the on-line decision problem," *Games and Economic Behavior*, vol. 29, no. 1-2, 1999.

[32] E. Catto, "Box2D, a 2D physics engine for games," http://box2d.org, 2011.

[33] J. J. Kuffner, Jr. and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *ICRA*, 2000.

[1] We use the focused algorithm within STRIPStream, and it solves the task in 20-40 seconds for a range of different arrangements of objects.