STAMP: Differentiable Task and Motion Planning via Stein Variational Gradient Descent

Yewon Lee, Andrew Z. Li, Philip Huang, Eric Heiden, Krishna Murthy Jatavallabhula, Fabian Damken, Kevin Smith, Derek Nowrouzezahrai, Fabio Ramos, *Member, IEEE*, and Florian Shkurti

***Abstract*—Planning for sequential robotics tasks often requires integrated symbolic and geometric reasoning. TAMP algorithms typically solve these problems by performing a tree search over high-level task sequences while checking for kinematic and dynamic feasibility. This can be inefficient because, typically, candidate task plans resulting from the tree search ignore geometric information. This often leads to motion planning failures that require expensive backtracking steps to find alternative task plans. We propose a novel approach to TAMP called Stein Task and Motion Planning (STAMP) that relaxes the hybrid optimization problem into a continuous domain. This allows us to leverage gradients from differentiable physics simulation to fully optimize discrete and continuous plan parameters for TAMP. In particular, we solve the optimization problem using a gradient-based variational inference algorithm called Stein Variational Gradient Descent. This allows us to find a distribution of solutions within a single optimization run. Furthermore, we use an off-the-shelf differentiable physics simulator that is parallelized on the GPU to run parallelized inference over diverse plan parameters. We demonstrate our method on a variety of problems and show that it can find multiple diverse plans in a single optimization run while also being significantly faster than existing approaches.**

***Index Terms*—Task and motion planning, probabilistic inference.**

I. Introduction

TASK and Motion Planning (TAMP) is central to many sequential decision-making problems in robotics, which often require integrated logical and geometric reasoning to generate a feasible symbolic action and motion plan that achieves a particular goal [1]. In this letter, we present a novel algorithm called STAMP, which uses SVGD [2], [3], a variational inference method, to efficiently generate a distribution of optimal solutions upon convergence. Unlike existing methods, we transform TAMP problems, which operate over both discrete symbolic variables and continuous motion variables, into the continuous domain. This allows us to run gradient-based inference using SVGD and differentiable physics simulation to generate a diversity of plans.

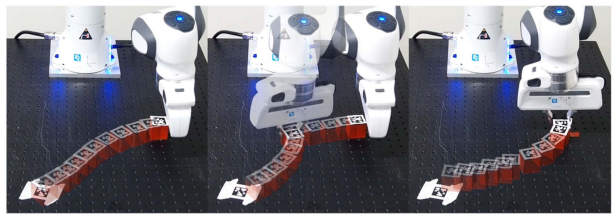


Fig. 1. Demonstrations of the top-3 block-pushing plans with up to two tasks ($K = 2$) found by STAMP on a Franka manipulator. Task plans from left to right: (a) push the East side; (b) push the East side, then the North side; (c) push the North side. STAMP found all three solutions in parallel in one run.

Prior works such as [4], [5], [6], [7], [8] solve TAMP problems by performing a tree search over discrete logical plans and integrating this with motion optimization and feasibility checking. By leveraging gradient information, STAMP forgoes the need to conduct a computationally expensive tree search that might involve backtracking and might be hard to parallelize. Instead, STAMP infers the relaxed logical action sequences jointly with continuous motion plans, without a tree search.

Further, by solving a Bayesian inference problem over the search space and utilizing GPU parallelization, STAMP conducts a parallelized optimization over multiple logical and geometric plans at once. As a result, it produces large, diverse plan sets that are crucial in downstream tasks with replanning, unknown user preferences, or uncertain environments. While several diverse planning methods have been developed for purely symbolic planning [9], [10], most TAMP algorithms do not explicitly solve for multiple plans and suffer from an exploration-versus-computation time trade-off.

Why would we need multiple solutions if only one will eventually be executed? One reason is that having access to a diverse set of solutions provides added flexibility in selecting a feasible plan based on criteria that were not initially considered. Second, the push for diverse solutions might lead to unexpected, but feasible plans. Third, when TAMP is used as a generating process for training data for imitation learning [11], a diverse set of solutions

Received 13 November 2024; accepted 14 March 2025. Date of publication 16 April 2025; date of current version 5 May 2025. This article was recommended for publication by Associate Editor M. Burke and Editor A. Faust upon evaluation of the reviewers' comments. This work of Yewon Lee was supported in part by the Google DeepMind Fellowship, in part byNSERC Canada Graduate Scholarship – Master's, and in part by Ontario Graduate Scholarship. *(Corresponding author: Yewon Lee.)*

Yewon Lee, Andrew Z. Li, and Florian Shkurti are with the University of Toronto, Toronto, ON M5S 1A1, Canada (e-mail: yewonlee@cs.toronto.edu).

Philip Huang is with the Carnegie Mellon University, Pittsburgh, PA 15213 USA.

Eric Heiden is with NVIDIA, Santa Clara, CA 95051 USA.

Krishna Murthy Jatavallabhula and Kevin Smith are with the Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Fabian Damken is with the University of Toronto, Toronto, ON M5S 1A1, Canada, and also with the Technische Universität Darmstadt, 64289 Darmstadt, Germany.

Derek Nowrouzezahrai is with the McGill University, Montreal, Quebec H3A 0G4, Canada.

Fabio Ramos is with NVIDIA, Santa Clara, CA 95051 USA, and also with the University of Sydney, Sydney, NSW 2050, Australia.

https://rvl.cs.toronto.edu/stamp.

This article has supplementary downloadable material available at https://doi.org/10.1109/LRA.2025.3561575, provided by the authors.

Digital Object Identifier 10.1109/LRA.2025.3561575

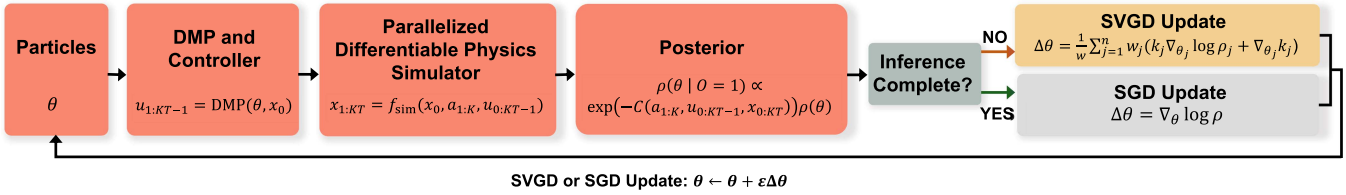


Fig. 2. Overview of STAMP algorithm pipeline. The particles θ represent the task and motion plan $\theta = [a_{1:K}, u_{0:KT-1}]$, or the task plan and a parametrization of the motion plan $\theta = [a_{1:K}, g_{1:K}]$ if Dynamic Movement Primitives dependent on goals $g_{1:K}$ are used (see Section IV-B). The resulting controls are passed through a differentiable physics simulator, and particles are updated in parallel using two phases of optimization: an inference phase of SVGD that balances loss minimization with diversification of particles followed by a finetuning phase of SGD so that particles can reach the local optima.

is preferable to learn policies that induce multi-modal trajectory distributions.

The main contributions of our work are (a) introducing a relaxation of the discrete symbolic actions and thus reformulating TAMP as an inference problem on continuous variables to avoid tree search; (b) solving the resulting problem with gradient-based SVGD inference updates using an off-the-shelf differentiable physics simulator; and (c) parallelizing the inference process on the GPU, so that multiple diverse plans can be found in one optimization run.

II. Related Work

Task and Motion Planning Guo et al. [12] categorize TAMP solvers into sampling-based methods [4], [13], [14], [15], [16], [17]; hierarchical methods [18], [19], [20]; constraint-based methods [21], [22], [23]; and optimization-based methods [5], [24], [25]. Our method falls under optimization-based methods, which find a complete task and motion plan that optimizes a predefined cost function.

Probabilistic Planning: Our work is inspired by the idea of approaching planning as inference [26]. Prior works such as [19], [27], [28], [29], [30], [31], [32], [33], [34] have explored the intersection of probability and TAMP. Ha et al. [28] developed probabilistic Logic Geometric Programming (LGP) for solving TAMP in stochastic environments, Shah et al. [27] developed an anytime algorithm for TAMP in stochastic environments, and Kaelbling and Lozano-Pérez extended Hierarchical Planning in the Now [18] to handle current and future state uncertainty. STAMP's probabilistic interpretation is similar to [28], [35] in that we run inference over a posterior plan distribution, but differs in that our distribution is defined over both discrete and continuous plan parameters rather than over only continuous parameters. While many stochastic TAMP methods can be computationally expensive [12], our method runs efficient, gradient-based inference through parallelization.

Diverse Planning: Diverse or top-k symbolic planners like SYM-K and FORBID-K are used to produce sets of feasible task plans [9], [10]. Existing work in TAMP generate different logical plans by adapting diverse symbolic planners, which are iteratively updated with feasibility feedback from the motion planner [6], [8]. Ren et al. [8] rely on a top-k planner to generate a set of candidate logical plans, but only to efficiently find a single TAMP solution rather than a distribution of diverse plans. More similarly to TAMP, Ortiz et al. [6] seek to generate a set of plans based on a novelty criteria, but enforce task diversity by iteratively forbidding paths in the logical planner. In contrast, STAMP finds diverse plans by solving TAMP as an inference problem over plan parameters.

Differentiable Physics Simulation & TAMP: Differentiable physics simulators [36], [37], [38], [39], [40], [41], [42], [43] solve a mathematical model of a physical system while allowing the computation of the first-order gradient of the output directly with respect to the parameters or inputs of the system. They have been used to optimize trajectories [39], controls [44], or policies [45]; and for system identification [40], [46]. Toussaint et al. [47] have used differentiable simulation within LGP for sequential manipulation tasks by leveraging simulation gradients for optimization at the path-level. In contrast, STAMP uses simulation gradients to optimize both symbolic and geometric parameters.

Envall et al. [48] used gradient-based optimization for task assignment and motion planning. Their problem formulation allows task assignments to emerge implicitly in the solution. In contrast, we optimize the task plan explicitly through continuous relaxations, use gradient-based inference to solve TAMP, and obtain gradients from differentiable simulation.

III. Preliminaries

A. Stein Variational Gradient Descent

SVGD is a variational inference algorithm that uses particles to fit a target distribution. Particles are sampled randomly at initialization and updated iteratively until convergence, using gradients of the target distribution with respect to each particle [2]. SVGD is fast, parallelizable, and able to fit both continuous [2] and discrete [3] distributions.

1) SVGD for Continuous Distributions [2]: Given a target distribution $p(\theta), \theta \in \mathbb{R}^d$, a randomly initialized set of particles $\{\theta_i\}_{i=1}^n$, a positive definite kernel $k(\theta, \theta')$, and step size ϵ, SVGD iteratively applies the following update rule on $\{\theta_i\}_{i=1}^n$ to approximate the target distribution $p(\theta)$ (where $k_{ji} = k(\theta_j, \theta_i)$ for brevity):

$$\theta_i \leftarrow \theta_i + \frac{\epsilon}{n}\sum_{j=1}^{n}\Big[\underbrace{\nabla_{\theta_j}\log p(\theta_j)k_{ji}}_{\text{(A)}} + \underbrace{\nabla_{\theta_j}k_{ji}}_{\text{(B)}}\Big] \tag{1}$$

Term (A), which is a kernel-weighted gradient, encourages the particles to converge towards high-density regions in the target distribution. Term (B) induces a "repulsive force" that prevents all particles from collapsing to a maximum *a posteriori* solution, i.e., it encourages exploration while searching over continuous parameters. This property allows STAMP to find multiple diverse solutions in parallel.

2) SVGD for Discrete Distributions [3]: Given a target distribution $p_*(z), z \in \mathcal{Z}$ on a discrete set $\mathcal{Z}$, DSVGD introduces

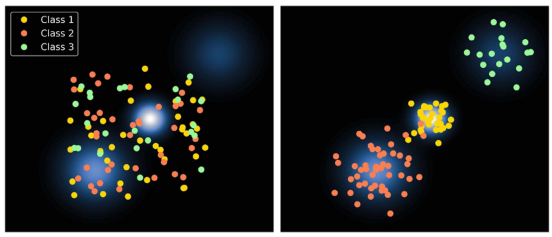


Fig. 3. Fitting a Gaussian mixture with discrete-and-continuous SVGD. Left: random initialization of particles. Right: particles after 500 updates. Note that same-color particles have been assigned to the same mode.

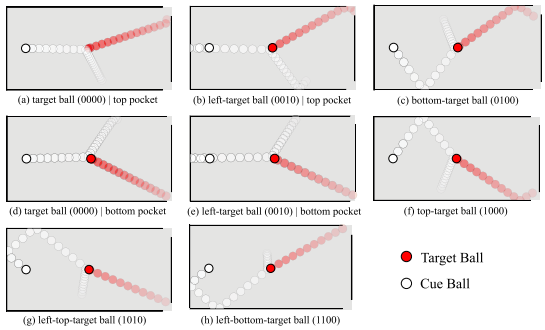


Fig. 4. Sample solutions STAMP found for the billiards problem. Subcaptions indicate which walls the cue ball hits before hitting the target ball. The $1^{st}/2^{nd}/3^{rd}/4^{th}$ digit correspond to hitting the top/bottom/left/right wall. A 1 means the wall is hit during the shot; a 0 means that it is not. The caption also indicates in which pocket the target ball is shot.

relaxations to $\mathcal{Z}$ that reformulates discrete inference as inference on a continuous domain. In our case, z denotes a symbolic/discrete action. To do the relaxation, we construct a differentiable and continuous surrogate distribution $\rho(\theta), \theta \in \mathbb{R}^d$, that approximates the discrete distribution $p_*(z)$. Crucially, a map $\Gamma : \mathbb{R}^d \to \mathcal{Z}$ is defined such that it divides an arbitrary base distribution $p_0(\theta), \theta \in \mathbb{R}^d$ (e.g., a Gaussian or uniform distribution) into K partitions with equal probability. That is,

$$\int_{\mathbb{R}^d} p_0(\theta)\, \mathbb{I}\left[z_i = \Gamma(\theta)\right]\, \mathrm{d}\theta = 1/K \tag{2}$$

Then, the surrogate distribution $\rho(\theta)$ can be defined as $\rho(\theta) \propto p_0(\theta)\tilde{p}_*(\tilde{\Gamma}(\theta))$ [3], where $\tilde{p}_*(\theta), \theta \in \mathbb{R}^d$ simply denotes $p_*(z)$, $z \in \mathcal{Z}$ defined on the continuous domain. $\tilde{\Gamma}(\theta)$ is a smooth relaxation of $\Gamma(\theta)$; for instance, $\tilde{\Gamma}(\theta) = \mathrm{softmax}(\theta)$ is a smooth relaxation of $\Gamma(\theta) = \max(\theta)$. After initializing particles $\{\theta_i\}_{i=1}^N$ *defined on the continuous domain*, DSVGD uses the surrogate to update $\{\theta_i\}_{i=1}^N$ via $\theta_i \leftarrow \theta_i + \epsilon\Delta\theta_i$, where $k_{ji} = k(\theta_j, \theta_i)$, $w = \sum_i w_i$, $\rho_j = \rho(\theta_j)$, and:

$$\Delta\theta_i = \sum_{j=1}^{n} \frac{w_j}{w}\left(\nabla_{\theta_j} \log \rho_j k_{ji} + \nabla_{\theta_j} k_{ji}\right),\ w_j = \frac{\tilde{p}_*\left(\tilde{\Gamma}(\theta_j)\right)}{p_*\left(\Gamma(\theta_j)\right)} \tag{3}$$

Intuitively, weights w_j correct for the bias introduced by employing the surrogate distribution in place of the discrete distribution. We use the RBF kernel for $k(\cdot)$, which is a popular choice in the SVGD literature. Post-convergence, the discrete counterpart of each particle can be recovered by evaluating $\{z_i = \Gamma(\theta_i)\}_{i=1}^N$.

B. Differentiable Physics Simulation

Physics simulators roll out the future states of a system given its initial pose x_0, control inputs $u_{0:T-1}$, and in some cases, the discrete actions $a_{0:T-1}$. We represent simulator rollouts as $x_{1:T} = f_{\text{sim}}(x_0, a_{0:T-1}, u_{0:T-1})$. Simulating future rollouts involves solving the dynamics equation

$$M\ddot{x} = J^\top F(x, \dot{x}) + C(x, \dot{x}) + \tau(x, \dot{x}, u), \tag{4}$$

where F denotes the external force, C the Coriolis force, and τ the joint actuations. The Warp simulator [41] used in our letter solves (4) for future states via time integration based on Semi-Implicit Euler [49] or XPBD [50], [51] schemes, while resolving contacts using a spring-based non-penetrative model [45], [52] and enforcing joint limits for articulated bodies using the spring model in [45]. Forward simulations in Warp can be parallelized on the GPU. As a differentiable physics simulator, Warp can also compute auto-differentiation gradients of future states and parameters with respect to past states and parameters. In other words, given a loss $\mathcal{L}$ defined over the final or intermediate states, gradients can be backpropagated through the entire trajectory with respect to simulation and plan parameters. As we will explain in following sections, we leverage the differentiability of Warp to compute the gradients $\nabla_{\theta_j} \log p(\theta_j)$ in term (A) of (1).

IV. Our Method: Stein Task and Motion Planning

STAMP solves TAMP as a variational inference problem over discrete symbolic actions and continuous motion plans, conditioned on them being optimal. An analogy to this is trying to sample particles from a known mixture of Gaussians, in which each Gaussian has a distinct class. SVGD must move the particles towards high-density regions in the Gaussian (thus learning the continuous parameters) and also learn the correct classes. In TAMP, the Gaussian mixture is analogous to a target distribution where higher likelihood is assigned to optimal solutions, the discrete class is analogous to the discrete task plan, and the continuous parameters are analogous to the motion plan. Fig. 3 illustrates how, after SVGD, particles with the same color usually belong to the same Gaussian.

A. Problem Formulation

Given a problem domain, we are interested in sampling optimal symbolic/discrete actions $z_{1:K} \in \mathcal{Z}^K$ and continuous controls $u_{0:KT-1} \in \mathbb{R}^{KT}$ from the posterior

$$p\left(z_{1:K}, u_{0:KT-1} \mid O = 1\right) \tag{5}$$

where $O \in \{0, 1\}$ indicates optimality of the plan; K is the number of symbolic action sequences; T is the number of timesteps by which we discretize each action sequence; and $\mathcal{Z}$ is a discrete set of all m possible symbolic actions that can be executed in the domain (i.e., $|\mathcal{Z}| = m$). $z_{1:K}$ and $u_{0:KT-1}$ fully parameterize a task and motion plan, since they can be input to a physics simulator f_{sim} along with the initial state x_0 to roll out the system's state at every timestep, that is, $x_{1:KT} = f_{\text{sim}}(x_0, z_{1:K}, u_{0:KT-1})$[1].

[1]We later introduce $a_{1:K}$, a continuous relaxation of $z_{1:K}$. The forward simulation can also be rolled out using $a_{1:K}$ as $x_{1:KT} = f_{\text{sim}}(x_0, a_{1:K}, u_{0:KT-1})$, which will ensure gradients can be taken with respect to $a_{1:K}$. This is important as $z_{1:K}$ is discrete.

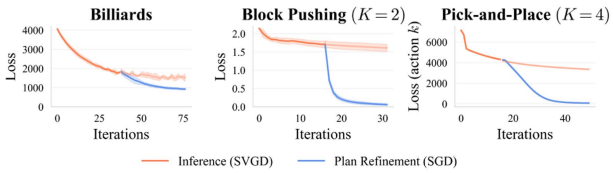


Fig. 5. Evolution of mean cost over all particles averaged across 5 runs. The red curve shows the mean cost when running SVGD inference, while the blue curve shows the mean cost after switching to SGD plan refinement after SVGD converges. Switching from SVGD to SGD allows STAMP to refine the plans towards exact optima. This is shown by the further reduction in mean cost that SGD achieves as compared to running SVGD only.

We use SVGD to sample optimal plans from $p(z_{1:K}, u_{0:KT-1} \mid O=1)$. As p is defined over both continuous and discrete domains, to use SVGD's gradient-based update rule, we follow Han et al. [3]'s approach and construct a differentiable surrogate distribution for p defined over a continuous domain. We denote this surrogate by ρ.

To construct the surrogate, we first relax the discrete variables by introducing a map from the real domain to $\mathcal{Z}^K$ that evenly partitions some base distribution p_0. For any problem domain with m symbolic actions, we define $\mathcal{Z} = \{e_i\}_{i=1}^m$ as a collection of m-dimensional one-hot vectors[2]. Then, $z_{1:K} \in \mathcal{Z}^K$, implying that we commit to one action (e_i) in each of the K action phases. Further, we use a uniform distribution for the base distribution p_0. Given this formulation, we can define our map[3] $\Gamma : \mathbb{R}^{mK} \to \mathcal{Z}^K$ and its differentiable surrogate $\tilde{\Gamma}$ as the following:

$$\Gamma(a_{1:K}) = \left[\max\{a_1\} \ldots \max\{a_K\}\right]^\top = z_{1:K}$$
$$\tilde{\Gamma}(a_{1:K}) = \left[\text{softmax}\{a_1\} \ldots \text{softmax}\{a_K\}\right]^\top = \tilde{z}_{1:K}. \quad (6)$$

The above map partitions p_0 evenly when p_0 is the uniform distribution. By constructing the above map, we can run inference over purely continuous variables $a_{1:K} \in \mathbb{R}^{mK}$ and $u_{0:KT-1} \in \mathbb{R}^{KT-1}$ and recover the discrete plan parameters post-inference via $z_{1:K} = \Gamma(a_{1:K})$. We denote $a_{1:K} \in \mathbb{R}^{mK}$ and $u_{0:KT-1} \in \mathbb{R}^{KT-1}$ collectively as a particle, that is, $\theta = [a_{1:K}, u_{0:KT-1}]^\top$, and randomly initialize $\{\theta_i\}_{i=1}^n$ to run SVGD inference. The target distribution we aim to infer is a differentiable surrogate ρ of the posterior p defined as:

$$\rho\left(a_{1:K}, u_{0:KT-1} \mid O=1\right) \quad (7)$$
$$\propto p_0(a_{1:K})\, p\left(\tilde{\Gamma}(a_{1:K}), u_{0:KT-1} \mid O=1\right). \quad (8)$$

In practice, because $p_0(a_{1:K})$ can be treated as a constant by making its boundary arbitrarily large, we simply remove p_0 from the expression above.

Note that it is not necessary to normalize the ρ as the normalization constant vanishes by taking the gradient of $\log \rho$ in the SVGD updates. We now quantify the surrogate of the posterior distribution. We begin by applying Bayes' rule and obtain the following factorization of $\rho(\theta \mid O=1)$:

$$\rho(\theta \mid O=1) \propto p(O=1 \mid \theta)\, p(\theta) \quad (9)$$

[2]The ith element of e_i is 1 and all other elements are 0.

[3]Given $a_k \in \mathbb{R}^m$, $\max\{a_k\}$ returns a m-dimensional one-hot encoding e_i iff the ith element of a_k is the larger than all other elements.

Since the likelihood function is synonymous with a plan's optimality, we formulate the likelihood via the total cost C associated with the relaxed task and motion plan θ:

$$p(O=1|\theta) \propto \exp -C(\theta, x_{0:KT}) \quad (10)$$

Meanwhile, the prior $p(\theta)$ is defined over θ only and as such, we use it to impose constraints on the plan parameters (e.g., kinematic constraints on robot joints).

Related to the Logic Geometric Programming (LGP) [5], [28], [47], [53], [54] framework, which optimizes a cost subject to constraint functions that activate or deactivate when kinematic/logical action transition events occur, $C(\theta)$ in our method is a sum of *relaxed* versions of both the cost and constraint functions. Here, "relaxed" refers to removing the cost and constraints' dependence on the discrete logical variables through our relaxations, and defining costs/constraint functions that are differentiable w.r.t. our plan parameters θ.

B. Problem Reduction Using Motion Primitives

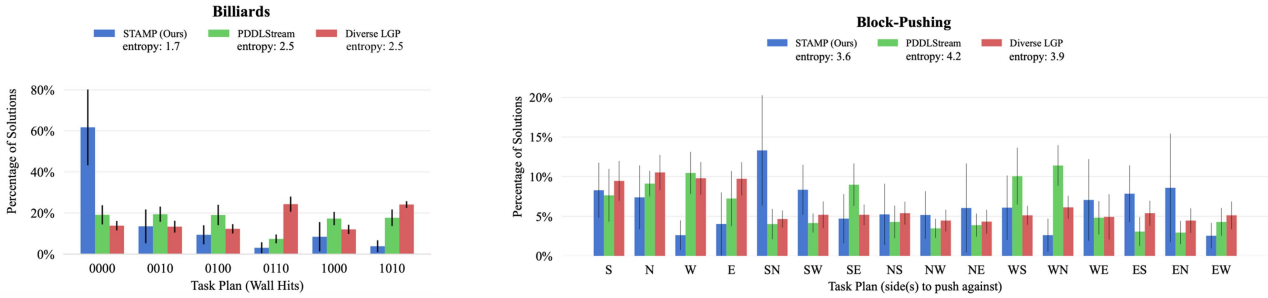
Rather than running inference over $\theta = [a_{1:K}, u_{0:KT-1}]^\top$, we can run inference over a smaller number of dimensions by *parametrizing* the continuous motion plan $u_{0:KT-1}$. There are many ways to do this, such as using splines to represent the motion plan, but we use goal-conditioned dynamic motion primitives (DMP). DMPs [55] model complex movements via a system of differential equations:

$$\tau \dot{v} = K(g-x) - Dv - K(g-x_0)s + Kf(s)$$
$$\tau \dot{x} = vf(s) = \frac{\sum_i w_i \psi_i(s) s}{\sum_i \psi_i(s)}$$
$$\tau \dot{s} = -\alpha s \quad (11)$$

Given a demonstration dataset, we "train" a DMP offline by fitting the (linear) parameters w_i of the DMP in (11) via solving linear least squares. At runtime, we use the offline-trained weights w_i to generate new motions simply by specifying the robot's initial state x_0 and goal state g, and integrating the above system of equations for an entire trajectory. In other words, we can redefine the particle as $\theta = [a_{1:K}, g_{1:K}]^\top$, where $g_{1:K} \in \mathbb{R}^{Kd}$ denotes goal poses for the system after executing action k, and d is the DOF of the system. With just g_k, we

Algorithm 1: Stein TAMP (STAMP).

let: *step size* = ϵ, *phase* = *SVGD*
initialize: n particles (candidate task and motion plans) $\{\theta_i\}_{i=1}^n$ randomly, where $\theta_i = [a_{1:K}, u_{0:KT\ \ 1}]_i$
while *not converged* **do**
  $[x_{1:KT}]_i = f_{\text{sim}}(x_0, \theta_i)$
  $\rho(\theta_i|O{=}1) \propto \exp\{-C(\theta_i, [x_{0:KT}]_i)\}\rho(\theta_i)$
  if *phase* = *SVGD* **then**
    $\Delta\theta_i = \frac{1}{w}\sum_{j=1}^n w_j\left[\nabla_{\theta_j} \log \rho_j k_{ji} + \nabla_{\theta_j} k_{ji}\right]\ \forall i$
    if *(absolute change in* $\sum_i \log \rho_i$*)* $< \delta$ **then**
      phase = *SGD*
  else if *phase* = *SGD* **then**
    $\Delta\theta_i = \nabla_{\theta_i} \log \rho(\theta_i \mid O=1)\ \forall i$
  $\theta_i \leftarrow \theta_i + \epsilon\Delta\theta_i\ \forall i$ in parallel



(a) STAMP found 104 solutions in (6.94 ± 0.06) s. Diverse LGP and PDDLStream are invoked 104 times, for a total of (780 ± 9) s and (595 ± 8) s, respectively.

(b) $K = 2$. STAMP found 825 solutions in (12.92 ± 0.04) s. Diverse LGP and PDDLStream are invoked 150 times, for a total of (3390 ± 90) s and (3330 ± 1) s, respectively.

Fig. 6. Distribution of plans found by STAMP, Diverse LGP, and PDDLStream. The vertical axis shows the number of times a particular symbolic plan was discovered normalized by the total number of plans found. Results are averages across 10 seeds; error bars represent standard deviation. STAMP's distribution is of lower entropy, because as a probabilistic inference method, more of its particles converge to plans with higher likelihood, whereas the baselines, which were modified to ignore previously-found plans, find plans more uniformly. Without this modification, the baselines would have 0 entropy.

recover the full trajectory $x_{0:KT-1}$ by integrating (11) and computing the controls $u_{0:KT-1}$ using a PD controller. This new inference problem over $[a_{1:K}, g_{1:K}]^\top$ is more tractable than the original inference problem over $[a_{1:K}, u_{0:KT-1}]^\top$ due to the reduced dimensionality. We use DMPs for our block-pushing and pick-and-place experiments, which have high DOF and planning horizon.

C. STAMP Algorithm

Given a problem domain with cost $C(\theta)$ and n randomly initialized particles $\theta = \{\theta_i\}_{i=1}^n$, STAMP finds a distribution of optimal solutions to the inference problem $\rho(\theta|O = 1)$ by running SVGD inference until convergence and subsequently refining the plans via stochastic gradient descent (SGD) updates. During SVGD, each of the n plans (θ) are executed and simulated in Warp. After obtaining states $x_{0:KT} = f_{\text{sim}}(x_0, a_{1:K}, u_{0:KT-1})$ for each particle, the posterior distribution and its surrogate are computed using (5) and (7). Gradients of the log-posterior with respect to $\{\theta_i\}_{i=1}^n$ are obtained via auto-differentiation in Warp, and these gradients are used to update each candidate plan θ_i via update rule (3). Once SVGD converges, *i.e.,* the absolute change in $\sum_i \log \rho_i$ in two consecutive iterations is less than some $\delta > 0$ (δ is a hyperparameter that we find empirically), we switch to SGD for all particles. During SGD, the above steps are repeated except for the update rule, which switches to $\theta_i \leftarrow \theta_i + \epsilon \nabla_{\theta_i} \log \rho_i$. Switching to SGD after SVGD allows us to finetune our plans, as the repulsive term $\nabla_\theta k(\theta, \theta')$ in SVGD can push the plans away from optima, but with SGD the plans are optimized to reach them. All of the above computations (physics simulations, log posterior evaluations, and gradient computations for all particles) are run in parallel on the GPU, making STAMP highly efficient. The pipeline and pseudocode are in Fig. 2 and Algorithm 1.

V. Evaluation

We run STAMP on three problems: billiards, block-pushing, and pick-and-place. We benchmark STAMP against two baselines that build upon PDDLStream [4] and Diverse LGP [6], with K fixed to the same number as STAMP. In block-pushing, we integrate the original baselines' task planner with a motion planning pipeline similar to STAMP's (we optimize intermediate goal poses of a DMP-generated trajectory via SGD). In pick-and-place, we benchmark against PDDLStream, and again use DMP-generated trajectories but without optimizing the intermediate goal poses (we assume that the first sampled motion plan is always correct), which provides a lower-bound estimate of PDDLStream's runtime. We run our experiments on the NVIDIA GeForce RTX 2080 Ti GPU and Intel Core™i7-9700K Processor.

A. Problem Environments and Their Algorithmic Setup

1) Billiards: The goal is to optimize the initial velocity u_0 on the cue ball that sends the target ball into one of the pockets in Fig. 4. This requires planning on the continuous domain ($u_0 \in \mathbb{R}^2$) and discrete domain (the set of walls we wish for the cue ball to hit before colliding with the target ball). We define our particles as $\theta = [u_0, z]$ where $z = [z_1, z_2, z_3, z_4] \in \{0, 1\}^4$, which indicates which of the four walls the cue ball bounces off of[4]. The cost function C is a weighted sum of the target loss L_{target} and aim loss L_{aim}, which are both squared L_2 distance functions which are respectively 0 when the cue ball ends up in either of the two pockets and if the cue ball comes in contact with the target ball at any point in its trajectory:

$$C(\theta) = \beta_{\text{aim}} L_{\text{aim}} + \beta_{\text{target}} L_{\text{target}} \tag{12}$$

Hyperparameters $\beta_{\text{target}}, \beta_{\text{aim}} > 0$ are the loss' respective coefficients.

2) Block-Pushing: The goal is to push the block in Fig. 1 towards the goal region. The symbolic plan is the sequence of sides (north, east, south, west) to push against. Assuming *a priori* that up to K action sequences can be committed, we define the particles as $\theta = \left[a_1, \ldots, a_K, g_1, \ldots, g_K\right]^\top$, where $a_k \in \mathbb{R}^4$ are the relaxed symbolic variables and $g_k = [g_k^x, g_k^y, g_k^\phi] \in \mathbb{R}^3$ denote individual goal poses after executing each action. Given the goals, we use a DMP trained on 47 pushing demonstrations to obtain trajectories and control inputs, by integrating (11). The discrete task variable for the kth action can be recovered via (6). The cost is the weighted sum of the target loss L_{target} and trajectory loss L_{traj}. L_{target} is a squared L_2 norm between the

[4] $z_i = 1$ if the cue ball hits wall i, and it is 0 otherwise.

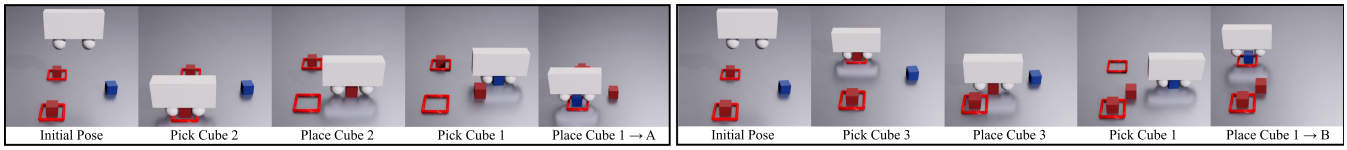


Fig. 7. Solutions STAMP found simultaneously for the pick and place experiment, with $K = 4$. The goal is to place the blue cube into one of the two targets on the left, which are occluded by red cubes. The solution on the left removes the bottom red cube to place the blue cube into the target. The solution on the right removes the top red cube to place the blue cube into the target.

cube and the target at the final time-step. L_{traj} is the L_1 distance traveled by the cube, which penalizes indirect paths to the target.

$$C(\theta) = \beta_{\text{target}} L_{\text{target}}\left(x_{KT}\right) + \beta_{\text{traj}} L_{\text{traj}}\left(g^{x,y}_{1:K}\right) \tag{13}$$

3) Pick-and-Place: Using an end effector, the goal is to pick and place blocks 1, 2, and 3 such that block 1 ends up in one of two targets as shown in Fig. 7. Long horizon reasoning is required in this problem, as blocks 2 and 3 (colored in red) occlude the targets; hence, either block 2 or 3 have to be moved out of the way before block 1 (blue) can be placed onto one of the targets. We express our goal as

$$\begin{aligned}&\left((\texttt{cube1} \in \mathcal{A}) \vee (\texttt{cube1} \in \mathcal{B})\right)\\ &\wedge \neg\texttt{onTop}(\texttt{cube1}, \texttt{cube2}) \wedge \neg\texttt{onTop}(\texttt{cube1}, \texttt{cube3}).\end{aligned} \tag{14}$$

Symbolic actions are $\mathcal{Z} = \{\texttt{pick}(c), \texttt{place}(c)\}\ \forall c \in \mathcal{C} = \{\texttt{cube1}, \texttt{cube2}, \texttt{cube3}\}$. We use the relaxation introduced in (6) to differentiably optimize the symbolic actions. Using this relaxation, particles are defined as $\theta = \left[a_1, \ldots, a_K, g_1, \ldots, g_K\right]^\top$, where $a_{1:K}$ represent which K-length sequence of symbolic actions are executed, and $g_{1:K}$ are the intermediate goal poses of the end effector after executing each action. We use a DMP trained on 10k autogenerated demonstrations to recover the end effector's full trajectory from $g_{1:K}$.

Unlike in previous examples, the symbolic actions have preconditions and postconditions that constitute constraints in our problem, as they must be satisfied before and after executing them. $\texttt{pick}(c)$, for instance, requires as preconditions that the end effector is not holding onto anything and that cube c is graspable; further, it requires as a postcondition that the end effector is holding onto c. For each pre/postcondition, we construct differentiable loss functions $L_{\texttt{pre}(\cdot)}, L_{\texttt{post}(\cdot)} \geq 0$ that are minimal when the condition is met[5]. The constraint associated with action $z \in \mathcal{Z}$ is then expressed as the γ_i- and γ_j-weighted sum of its precondition losses and postcondition losses, where $\gamma_i, \gamma_j > 0$ are scalar hyperparameters:

$$L_z(x_{t_0:t_T}) = \sum_{i \in \texttt{pre}(z)} \gamma_i L_i(x_{t_0}, g) + \sum_{j \in \texttt{post}(z)} \gamma_j L_j(x_{t_T}) \tag{15}$$

We optimize towards the logical goal stated in (14) by defining the total target loss as the sum of each cube's individual target loss weighted by a soft indicator function ω_c that gets softly activated whenever the cube is held by the end effector, and deactivated when it is not. The target loss of cube c, $L_{\text{target},c}$, and its gradient will dominate the optimization if c is held by the gripper, which is a necessary condition for placing c in its target.

$$L_{\text{target}}(x_{t_0}, x_{t_T}) = \sum_{c \in \mathcal{C}} \omega_c(x_{t_0}) \cdot L_{\text{target},c}(x_{t_T}). \tag{16}$$

[5]A comprehensive description of pre/postcondition loss definitions can be found in Appendix XII of the letters posted on our project website: https://rvl.cs.toronto.edu/stamp.

TABLE I
STAMP'S RUNTIME VS. BASELINES

	Method	Time / Solution (s)	Runtime (s)
Billiards	Diverse LGP	7.5 ± 0.9	7.5 ± 0.9
	PDDLStream	5.7 ± 0.8	5.7 ± 0.8
	STAMP	0.068 ± 0.007	6.94 ± 0.06
Pusher	Diverse LGP	22.6 ± 7.6	22.6 ± 7.6
	PDDLStream	22.2 ± 0.1	22.2 ± 0.1
	STAMP	0.0157 ± 0.0004	12.92 ± 0.04
P-P	PDDLStream†	25.00 ± 0.01	325.3 ± 0.2
	STAMP†	29 ± 9	382 ± 1
	STAMP	160 ±50	2111 ± 8

Averaged across 10 seeds; P-P denotes "pick-and-place"; $K = 2$ for pusher; $K = 4$ for P-P; PDDLStream† implements PDDLStream but assumes that the first sampled motion plan is correct, thus is a lower-bound estimate of PDDLStream's runtime; STAMP† measures runtime excluding the simulation or gradient computation time.

TABLE II
RUNTIME VS. # PARTICLES

#Particles	Billiards (s)	Pusher, $K = 2$ (s)
700	6.30 ± 0.07	12.53 ± 0.05
900	6.55 ± 0.04	12.74 ± 0.03
1100	6.94 ± 0.06	12.92 ± 0.04
1300	7.71 ± 0.12	13.22 ± 0.06

Averaged across 10 seeds.

TABLE III
RUNTIME VS. PARTICLE DIM. (PUSHER)

Max. #Pushes (K)	Dim.	Runtime (s)
2	16	12.92 ± 0.04
3	24	12.93 ± 0.04
4	32	13.26 ± 0.03
5	40	13.57 ± 0.03

Averaged across 10 seeds; 1100 particles.

The target loss for cube 1 is minimal when cube 1 is placed in one of the two targets, while for cube 2 and 3, it is minimal when the cubes are *not* in the two targets.

Finally, the total cost function is constructed as a sum over all constraints $\mathbf{L}_z(x_{t_0:t_T}) = [L_z(x_{t_0:t_T}) \forall z \in \mathcal{Z}]^\top$ weighted by $\tilde{z}_k = \tilde{\Gamma}(a_k) \forall k = 1, \ldots, K$ and the target loss, which is computed after executing each action[6].

$$C(\theta) = \sum_{k=1}^{K} \left(\tilde{z}_k \cdot \mathbf{L}_z(x_{(k-1)T+1:kT}) + L_{\text{target}}(x_{kT})\right) \tag{17}$$

[6]Recall that K is the number of action sequences, a_k the component of the particle representing the task plan, and $\tilde{\Gamma}(\cdot)$ a composition of softmax operations.

The global optima of $C(\theta)$ are thus action sequences $z_{1:K} = \Gamma(a_{1:K})$ and intermediate goal poses that solve the goal while ensuring all pre/postconditions are met. Further, C is amenable to gradient-based methods like STAMP since it is differentiable w.r.t. both $a_{1:K}$ and $g_{1:K}$. Note, $C(\theta)$ is the sum of individual loss terms which only depend on trajectories within each of the K action sequences; i.e., it is of the form:

$$C(\theta) = \sum_{k=1}^{K} C_k(a_k, g_k, x_{(k-1)T+1:kT}). \tag{18}$$

As the loss functions within the sums only depend on short trajectories (e.g., $x_{(k-1)T+1:kT}$ as opposed to $x_{1:KT}$) and the task variable a_k, we can split the optimization into smaller chunks by defining the posterior using the 'inner' cost C_k, which prevents the need for taking gradients over the entire trajectory $x_{1:KT}$ and mitigates the danger of gradient explosion during optimization.

B. Experimental Results (Simulation)

We investigate the following questions:

(Q1) Can STAMP return a variety of plans to problems for which multiple solutions are possible?
(Q2) How does STAMP's runtime compare to search-based TAMP baselines?
(Q3) How does STAMP's runtime scale w.r.t. problem dimensionality and number of particles?
(Q4) Is SGD necessary in STAMP to find optimal plans?

1) Solution Diversity: STAMP finds a variety of solutions to all three problems; Figs. 1, 4, and 7 show sample solutions. While STAMP finds a distribution of solutions in a single run, baseline methods can only find one solution per run. Fig. 6(a) and (b) show a histogram of solutions found in one run compared to solutions found using baseline methods by invoking them multiple times.

2) Runtime Efficiency: STAMP produces large plan sets in similar or substantially less time than baselines (Table I). For pick-and-place, most of STAMP's runtime is spent on simulation and backward pass, so we exclude simulation and backward pass time from the runtime when comparing it to PDDLStream's lower-bound runtime estimate. One way to reduce STAMP's runtime may be to use a neural network to learn the physics and differentiate through the network, but we limit the scope of our work to using a differentiable physics simulator.

3) Scalability to Higher Dimensions and Greater Number of Particles: Tables II and III show the total runtime of our algorithm while varying the number of particles and particle dimensions. Increasing the number of particles and the dimensions of the particles has little effect on the total optimization time, which is consistent with expectations as our method is parallelized over the GPU[7].

4) SGD Plan Refinement: Pure SVGD inference results in slow or poor convergence, while running SGD post-SVGD inference results in better convergence (Fig. 5).

C. Experimental Results (Real Robot)

We demonstrate STAMP on a real robot system using a front-view camera, AprilTags [56] for pose estimation of cube(s) and target(s), and a Franka Emika Panda robotic arm. We input the detected poses into STAMP and track the resulting trajectory directly. We vary the cube and target positions across 2 distinct configurations for block-pushing, and 1 configuration for pick-and-place, which can be seen on our project website[8] and in our video submission, along with their top solutions. Fig. 1 shows the top solutions[9] found for one configuration of block-pushing. The lowest-cost solutions found by STAMP, shown in our videos, were physically realistic and thus successful in our real robot experiments. Some of the other solutions found by STAMP were not executable in the real world due to sim2real gaps such as frictional differences. In our problem formulation, our cost function did not take into account the sim2real gap (e.g., rotational infeasibility in block-pushing, which depends on friction forces), which is typical in TAMP methods. However, because our method finds multiple solutions in parallel, the end user has the flexibility of choosing the best solution using costs that weren't originally considered in the initial formulation, such as the sim2real gap.

VI. Conclusion

We presented STAMP, which formulates TAMP as a variational inference problem over discrete symbolic action and continuous motion parameters, and solves the inference problem using SVGD and gradients from differentiable simulation. We validated our approach on billiards, block-pushing, and pick-and-place problems, where STAMP discovered a diverse set of plans covering multiple different task sequences and motion plans. Through exploiting parallel gradient computation from a differentiable simulator, STAMP finds a variety of solutions in a single optimization run, and its runtime scales well to higher dimensions and more particles.

References

[1] C. R. Garrett et al., "Integrated task and motion planning," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 4, pp. 265–293, 2021.
[2] Q. Liu and D. Wang, "Stein variational gradient descent: A general purpose Bayesian inference algorithm," in *Proc. Adv. Neural Inf. Process. Syst.*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., Curran Associates, Inc., 2016, pp. 2378–2386.
[3] J. Han, F. Ding, X. Liu, L. Torresani, J. Peng, and Q. Liu, "Stein variational inference for discrete distributions," in *Proc. Int. Conf. Artif. Intell. Statist.*, PMLR, 2020, pp. 4563–4572.
[4] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2020, pp. 440–448.
[5] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 1930–1936.
[6] J. Ortiz-Haro, E. Karpas, M. Toussaint, and M. Katz, "Conflict-directed diverse planning for logic-geometric programming," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2022, pp. 279–287.
[7] T. Ren, G. Chalvatzaki, and J. Peters, "Extended task and motion planning of long-horizon robot manipulation," 2021, *arXiv:2103.05456*.
[8] T. Ren, G. Chalvatzaki, and J. Peters, "Extended tree search for robot task and motion planning," in *Proc. 2024 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2021.
[9] D. Speck, R. Mattmüller, and B. Nebel, "Symbolic top-k planning," in *Proc. 34th AAAI Conf. Artif. Intell.*, V. Conitzer and F. Sha, Eds., AAAI Press, 2020, pp. 9967–9974.

[7] GPU parallelization is made possible through the use of SVGD, a parallelizable inference algorithm, and the Warp simulator

[8] [Online]. Available: https://rvl.cs.toronto.edu/stamp

[9] The lowest-cost solutions found for the most popular 3 symbolic plans.

[10] M. Katz, S. Sohrabi, O. Udrea, and D. Winterer, "A novel iterative approach to top-k planning," in *Proc. 28th Int. Conf. Automated Plan. Scheduling*, AAAI Press, 2018, pp. 132–140.

[11] M. Dalal, A. Mandlekar, C. Garrett, A. Handa, R. Salakhutdinov, and D. Fox, "Imitating task and motion planning with visuomotor transformers," in *Proc. Conf. Robot Learn.*, 2023, pp. 2565–2593.

[12] H. Guo, F. Wu, Y. Qin, R. Li, K. Li, and K. Li, "Recent trends in task and motion planning for robotics: A survey," *ACM Comput. Surv.*, vol. 55, pp. 1–36, 2023.

[13] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *Int. J. Robot. Res.*, vol. 37, no. 1, pp. 104–136, 2018.

[14] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Sampling-based methods for factored task and motion planning," *Int. J. Robot. Res.*, vol. 37, no. 13/14, pp. 1796–1825, 2018.

[15] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Stripstream: Integrating symbolic planners and blackbox samplers," in *Proc. ICAPS Workshop Plan. Robot.*, 2018.

[16] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2020, pp. 440–448.

[17] W. Thomason and R. A. Knepper, "A unified sampling-based approach to integrated task and motion planning," in *Proc. Int. Symp. Robot. Res.*, Springer, 2019, pp. 773–788.

[18] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 1470–1477.

[19] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *Int. J. Robot. Res.*, vol. 32, no. 9/10, pp. 1194–1227, Aug. 2013.

[20] A. Suárez-Hernández, G. Alenyà, and C. Torras, "Interleaving hierarchical task planning and motion constraint testing for dual-arm manipulation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 4061–4066.

[21] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Proc. Robot. Sci. Syst. Conf.*, Ann Arbor, MI, USA, 2016, Art. no. 00052.

[22] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *Int. J. Robot. Res.*, vol. 37, no. 10, pp. 1134–1151, 2018.

[23] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Towards manipulation planning with temporal logic specifications," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 346–352.

[24] C. Zhang and J. A. Shah, "Co-optimizing task and motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 4750–4756.

[25] S. Saha and A. A. Julius, "Task and motion planning for manipulator arms with metric temporal logic specifications," *IEEE Robot. Automat. Lett.*, vol. 3, no. 1, pp. 379–386, Jan. 2018.

[26] M. Botvinick and M. Toussaint, "Planning as inference," *Trends Cogn. Sci.*, vol. 16, no. 10, pp. 485–488, 2012.

[27] N. Shah, D. K. Vasudevan, K. Kumar, P. Kamojjhala, and S. Srivastava, "Anytime integrated task and motion policies for stochastic environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 9285–9291.

[28] J.-S. Ha, D. Driess, and M. Toussaint, "A probabilistic framework for constrained manipulations and task and motion planning under uncertainty," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 6745–6751.

[29] I. A. Şucan and L. E. Kavraki, "Accounting for uncertainty in simultaneous task and motion planning using task motion multigraphs," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2012, pp. 4822–4828.

[30] W. Zhao and W. Chen, "Hierarchical POMDP planning for object manipulation in clutter," *Robot. Auton. Syst.*, vol. 139, 2021, Art. no. 103736.

[31] L. Kaelbling and T. Lozano-Perez, "Domain and plan representation for task and motion planning in uncertain domains," in *Proc. IROS Workshop Knowl. Representation Auton. Robots*, 2011.

[32] D. Hadfield-Menell, E. Groshev, R. Chitnis, and P. Abbeel, "Modular task and motion planning in belief space," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 4991–4998.

[33] M. Hou, T. X. Lin, H. Zhou, W. Zhang, C. R. Edwards, and F. Zhang, "Belief space partitioning for symbolic motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 8245–8251.

[34] C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, "Online replanning in belief space for partially observable task and motion problems," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 5678–5684.

[35] A. Lambert, A. Fishman, D. Fox, B. Boots, and F. Ramos, "Stein variational model predictive control," in *Proc. Conf. Robot Learn.*, 2020, pp. 1278–1297.

[36] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *Proc. 32nd Conf. Neural Inf. Process. Syst.*, 2018, pp. 7178–7189.

[37] Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin, "Scalable differentiable physics for learning and control," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 7847–7856.

[38] K. Werling, D. Omens, J. Lee, I. Exarchos, and K. Liu, "Fast and feature-complete differentiable physics engine for articulated rigid bodies with contact constraints," in *Proc. 17th Robot. Sci. Syst.*, 2021.

[39] T. A. Howell, S. Le Cleac'h, J. Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A differentiable simulator for robotics," 2022, *arXiv:2203.00806*.

[40] J. K. Murthy et al., "Gradsim: Differentiable simulation for system identification and visuomotor control," in *Proc. Int. Conf. Learn. Representations*, 2021.

[41] M. Macklin, "Warp: A high-performance python framework for GPU simulation and graphics," 2022. [Online]. Available: https://github.com/nvidia/warp

[42] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax—A differentiable physics engine for large scale rigid body simulation," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, 2021.

[43] Y. Hu et al., "DiffTaichi: Differentiable programming for physical simulation," in *Proc. Int. Conf. Learn. Representations*, 2020.

[44] E. Heiden, M. Macklin, Y. Narang, D. Fox, A. Garg, and F. Ramos, "Disect: A differentiable simulation engine for autonomous robotic cutting," in *Proc. Robot. Sci. Syst. Conf.*, 2021.

[45] J. Xu et al., "Accelerated policy learning with parallel differentiable simulation," in *Proc. Int. Conf. Learn. Representations*, 2022.

[46] E. Heiden, C. E. Denniston, D. Millard, F. Ramos, and G. S. Sukhatme, "Probabilistic inference of simulation parameters via parallel differentiable simulation," in *Proc. Int. Conf. Robot. Automat.*, 2022, pp. 3638–3645.

[47] M. Toussaint, K. Allen, K. Smith, and J. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Proc. 14th Robot. Sci. Syst. Conf.*, 2018.

[48] J. Envall, R. Poranne, and S. Coros, "Differentiable task assignment and motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2023, pp. 2049–2056.

[49] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, muJoCo, ODE and physX," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 4397–4404.

[50] M. Macklin, M. Müller, and N. Chentanez, "XPBD: Position-based simulation of compliant constrained dynamics," in *Proc. 9th Int. Conf. Motion Games*, 2016, pp. 49–54.

[51] M. Macklin et al., "Small steps in physics simulation," in *Proc. 18th Annu. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2019, pp. 1–7.

[52] J. Xu et al., "An end-to-end differentiable framework for contact-aware robot design," in *Proc. Robot. Sci. Syst. Conf.*, 2021.

[53] D. Driess, J.-S. Ha, M. Toussaint, and R. Tedrake, "Learning models as functionals of signed-distance fields for manipulation planning," in *Proc. 5th Conf. Robot Learn.*, A. Faust, D. Hsu, and G. Neumann, Eds., PMLR, 2022, pp. 245–255.

[54] M. Toussaint, J.-S. Ha, and D. Driess, "Describing physics for physical reasoning: Force-based sequential manipulation planning," *IEEE Robot. Automat. Lett.*, vol. 5, no. 4, pp. 6209–6216, Oct. 2020.

[55] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 763–768.

[56] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 3400–3407. [Online]. Available: https://ieeexplore.ieee.org/document/5979561