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

Yewon Lee[1], Philip Huang[2], Krishna Murthy Jatavallabhula[3], Andrew Z. Li[1], Fabian Damken[1,6], Eric Heiden[8], Kevin Smith[3], Derek Nowrouzezahrai[7], Fabio Ramos[8,9], Florian Shkurti[1]

***Abstract*—Planning for many manipulation tasks, such as using tools or assembling parts, often requires both symbolic and geometric reasoning. Task and Motion Planning (TAMP) algorithms typically solve these problems by conducting a tree search over high-level task sequences while checking for kinematic and dynamic feasibility. This can be inefficient as the width of the tree can grow exponentially with the number of possible actions and objects. In this paper, we propose a novel approach to task and motion planning (TAMP) that relaxes discrete-and-continuous TAMP problems into inference problems on a continuous domain. Our method, Stein Task and Motion Planning (STAMP) subsequently solves this new problem using a gradient-based variational inference algorithm called Stein Variational Gradient Descent, by obtaining gradients from a parallelized differentiable physics simulator. By introducing relaxations to the discrete variables, leveraging parallelization, and approaching TAMP as an Bayesian inference problem, our method is able to efficiently find multiple diverse plans in a single optimization run. We demonstrate our method on two TAMP problems and benchmark them against existing TAMP baselines.**

I. Introduction

TAMP for robotics is a family of algorithms that conduct integrated logical and geometric reasoning, resulting in a symbolic action plan and a motion plan that are feasible and solve a particular goal [1]. Prior works such as [2]–[4] ultimately solve TAMP problems by conducting a tree search over discrete logical plans and integrating this with motion optimization and feasibility checking. While these works have shown success in solving a number of difficult problems, including long-horizon planning with tools [4], such an approach to TAMP has several limitations. First, the search over discrete symbolic plans can become highly inefficient, because the width of the tree can grow exponentially with the number of objects and possible successor states. Second, many TAMP algorithms with the exception of stochastic TAMP methods such as [5], [6] are deterministic and produce a single plan, and thus have a limited ability to deal with uncertainty in partially observable settings or settings with stochastic dynamics. On the other hand, while stochastic TAMP methods have the ability to deal with uncertain observations and dynamics, they are often computationally expensive [7].

In this paper, we present a novel algorithm for TAMP called Stein Task and Motion Planning (STAMP) that is both efficient and able to produce a distribution of optimal plans.

[1]University of Toronto, [2]Carnegie Mellon University, [3]Massachusetts Institute of Technology, [6]Technische Universität Darmstadt, [7]McGill University, [8]NVIDIA, [9]University of Sydney

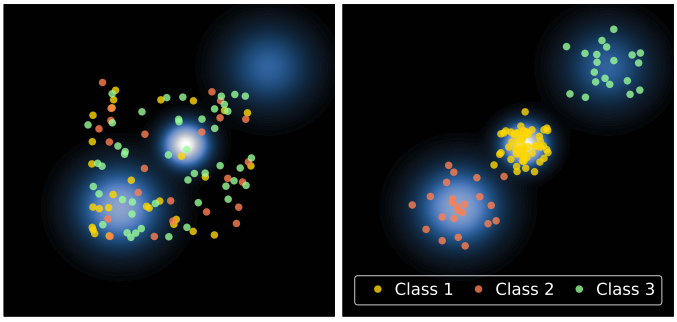


Figure 1. Fitting a Gaussian mixture with discrete-and-continuous SVGD. Left: random initialization of particles. Right: particles after 500 updates. In TAMP, classes 1-3 are analogous to the discrete symbolic plans while the particles' positions are analogous to the continuous motion plans. The target distribution is analogous to a posterior distribution over the task and motion plan conditioned on the plan being optimal. In other words, higher density regions in the posterior correspond to plans that better solve the problem.

Unlike traditional TAMP algorithms, STAMP solves TAMP in a *fully differentiable* manner, forgoing the need to conduct a computationally expensive search over discrete task plans integrated with motion planning. Further, it solves TAMP as a Bayesian inference problem that is parallelized on the GPU. These two factors combined allow STAMP to efficiently find a diverse set of plans in one algorithm run.

The main contributions of our work are (a) formulating TAMP as a Bayesian inference problem over continuous variables by relaxing the discrete symbolic actions into the continuous domain; (b) solving the new TAMP problem via a gradient-based inference algorithm, leveraging the end-to-end differentiability of differentiable physics simulators; and (c) parallelizing the inference process so that multiple diverse plans can be found in one optimization run. Our approach builds on top of a variational inference algorithm known as Stein variational gradient descent (SVGD) [8], [9] and uses a parallelized differentiable physics simulator [10]. These two ingredients allow us to approach TAMP as a probabilistic inference problem that can be solved using gradients, in parallel. In addition, we employ imitation learning to introduce action abstractions that reduce the high-dimensional variational inference problem over long trajectories to a low-dimensional inference problem. We compare our algorithm to logic-geometric programming (LGP) [4] and PDDLStream [3], two prominent TAMP baselines, and show that our method outperforms both algorithms in terms of the time required to generate a diversity of feasible plans.

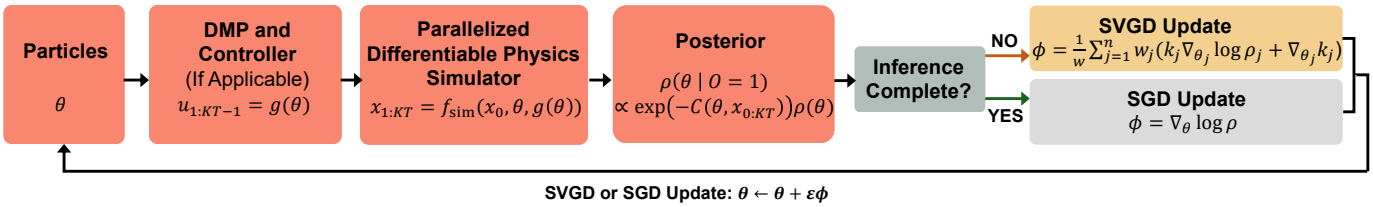


Figure 2. Overview of STAMP algorithm pipeline. The particles θ denote the action sequences and their parameters.

II. Related Work

There are many important prior works in the domain of task and motion planning. Garrett *et al.* [1] provide a good overview. A fundamental problem in TAMP is selecting real-valued parameters that define the mode of the motion bridging the motion planning problem with a high-level symbol. One solution introduced in PDDLStream [3] is to sample these parameters, such as an object's grasp pose and a trajectory's goal pose. Then, the problem can be reduced to a sequence of PDDL problems [11] and solved with a traditional symbolic planner. Our method is more closely related to a class of TAMP solvers known as LGP [4], [12]–[15], which optimize the motion plan directly on the geometric level to minimize an arbitrary cost function. Logical states then define various kinematic, dynamic, and mode transition constraints on the trajectories. There are several extensions to this line of work. Toussaint *et al.* [4] incorporate differentiable simulators to LGP for planning tool uses. Ha *et al.* [15] introduce probabilistic LGP for uncertain environments and Toussaint *et al.* [13] extend LGP with force-based reasoning to directly optimize physically plausible trajectories. While LGP uses branch-and-bound and tree search to solve the optimization problem, our method uses gradients from a differentiable simulator to directly optimize motions and (relaxed) symbolic variables.

Many recent works on differentiable simulators [10], [16]–[19] have shown that they can be used to optimize trajectories [20], controls [21], or policies [22] directly. Heiden *et al.* [23] also use Stein variational gradient descent [8] with gradients from differential simulators for system identification. Similar to our approach, their reason for using SVGD as the inference tool is its ability to find a globally optimal distribution from local gradients. Lambert *et al.* [24] show that SVGD can also solve short-horizon trajectory optimizations probabilistically for model predictive control. In contrast, we optimize both high-level symbolic and detailed motion plans with differentiable simulators.

Similar to our work, Envall *et al.* [25] developed a differentiable TAMP method. Their idea is to relax the kinematic switches and thereby transform TAMP into a continuous optimization problem that can be solved with gradients, using algorithms like Newton's Method. They do not explicitly solve for task assignments and instead allow them to emerge naturally from solving the continuous optimization problem. In contrast to their approach, we introduce relaxations to the symbolic tasks, optimize for the task plan directly, and approach TAMP as a variational inference problem that is solved with a gradient-based inference algorithm. Further, unlike their approach, our algorithm finds for multiple diverse plans in parallel.

III. Preliminaries

A. Stein Variational Gradient Descent

SVGD is a variational inference algorithm that uses samples (also called *particles*) to fit a target distribution. Running inference via SVGD involves randomly initializing a set of samples and then updating them iteratively, using gradients of the target distribution [8]. SVGD's iterative updates resemble gradient descent and are derived by minimizing the Kullback-Leibler (KL) divergence between the sample distribution and the target distribution within a function space [8]. SVGD is fast, parallelizable, and can be used to fit both continuous and discrete distributions. In the following, we will introduce these two flavors of SVGD.

1) SVGD for Continuous Distributions [8]: Given a target distribution $p(x)$, $x \in \mathbb{R}^d$, a randomly initialized set of samples $\{x_i\}_{i=1}^n$, a positive definite kernel $k(x, x')$, and a learning rate ϵ, SVGD iteratively applies the following update rule on $\{x_i\}_{i=1}^n$ to approximate the target distribution $p(x)$ (where $k_{ji} = k(x_j, x_i)$ for brevity):

$$x_i \leftarrow x_i + \frac{\epsilon}{n}\sum_{j=1}^{n}\left[\nabla_{x_j} \log p(x_j) k_{ji} + \nabla_{x_j} k_{ji}\right], \tag{1}$$

2) SVGD for Discrete Distributions [9]: The aim of SVGD for discrete distributions (DSVGD) is to approximate a target distribution $p_*(z)$ on a discrete set $\mathcal{Z} = \{z_1, \ldots, z_K\}$. To employ a similar gradient-based update rule to SVGD, DSVGD relies on the construction of a differentiable and continuous distribution $\rho(x)$, $x \in \mathbb{R}^d$, that mimics the discrete distribution $p_*(z)$. A crucial step in creating the surrogate, $\rho(x)$, from $p_*(z)$ is to find a map $\Gamma : \mathbb{R}^d \to \mathcal{Z}$ that divides a base distribution $p_0(x)$ (which can be any distribution, e.g., a Gaussian) into K partitions that have equal probability. That is,

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

Given this map, the authors of [9] suggest

$$\rho(x) \propto p_0(x) \tilde{p}_*\big(\tilde{\Gamma}(x)\big) \tag{3}$$

as one possibility of creating $\rho(x)$, where $\tilde{p}_*(\cdot)$ and $\tilde{\Gamma}(\cdot)$ are smooth and differentiable approximations of $p_*(\cdot)$ and $\Gamma(\cdot)$, respectively. DSVGD uses the surrogate to update the samples

via the following (where $k_{ji} = k(x_j, x_i)$ and $w = \sum_i w_i$ for brevity).

$$x_i \leftarrow x_i + \frac{\epsilon}{w} \sum_{j=1}^{n} w_j \big(\nabla_{x_j} \log \rho(x_j) k_{ji} + \nabla_{x_j} k_{ji} \big)$$
$$w_j = \frac{\tilde{p}_*\big(\tilde{\Gamma}(x_j)\big)}{p_*\big(\Gamma(x_j)\big)} \tag{4}$$

In the summation over the samples in equations (1) and (4), the first term encourages the samples to converge towards high-density regions in the target distribution, while the second term induces a "repulsive force" that discourages mode collapse by encouraging exploration.

B. Differentiable Physics Simulators

Differentiable physics simulators 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. A key step of our method relies on the ability to simulate the system and compute its gradients in parallel. Many existing differentiable simulators do not support this out-of-the-box, such as [17]–[20]. This is because they model the rigid-body contact problem as a complementary problem and solve it with optimization methods. While this may lead to more accurate gradients [26], optimization problems are harder to scale and solve in parallel in combination with existing auto-differentiation tools. In contrast, compliant methods model the contact surface as a soft "spring-damper" that produces an elastic force as penetration occurs [16]. These models can be more easily implemented in existing auto-differentiation frameworks and even support simulation in parallel [10], [27], [28]. Lastly, position-based dynamics (PBD) manipulates positions directly by projecting positions to valid locations that satisfy kinematic and contact constraints. Because the forward pass of positions can be implemented as differentiable operators, gradients can be computed easily and even in parallel. Two such differentiable simulators are Warp [10] and Brax [27] which both support parallelization using PBD.

C. Dynamic Motion Primitives

Dynamic Motion Primitives [29] generate trajectories from demonstrations. In practice, the framework models a movement with the following system of differential equations:

$$\begin{aligned} \tau \dot{v} &= K(g - x) - Dv - K(g - x_0)s + Kf(s) \\ \tau \dot{x} &= v \qquad\qquad f(s) = \frac{\sum_i w_i \psi_i(s) s}{\sum_i \psi_i(s)} \\ \tau \dot{s} &= -\alpha s \end{aligned} \tag{5}$$

Here, x and v are positions and velocities; x_0 and g are the start and goal positions, respectively; τ is a temporal scaling factor; K and D are the spring and damping constant, respectively; s is a phase variable that defines a "canonical system" in equation (5); and $\psi_i(s)$ are typically Gaussian basis functions with different centers and widths. Lastly, $f(s(t))$ is a non-linear function which can be learned to generate arbitrary trajectories from demonstrations. Time-discretized trajectories $x(t), v(t)$ are generated by integrating the system of differential equations in (5). Thus, learning a DMP from demonstrations can be formulated as a linear regression problem given recorded $x(t)$, $v(t)$, and $\dot{v}(t)$.

Adapting learned DMPs to new goals can be achieved by specifying task-specific parameters x_0, g, integrating $s(t)$, and computing $f(s)$, which drives the desired behaviour. Our work uses DMPs to generate motion primitives at varying speeds, initial positions, and goals from a small set of demonstrations. DMPs can also be extended to avoid collisions with obstacles by incorporating potential fields [30]–[32], a key requirement for many motion planning tasks.

Algorithm 1: Stein TAMP (STAMP)

Initialize: learning rate ϵ, phase = SVGD, n plans $\{\theta_i = [a_{1:K}, u_{0:KT-1}]_i = [a_i, u_i]\}_{i=1}^n$ randomly
while *not converged* **do**
  // simulate all $\{\theta_i\}_{i=1}^n$ in parallel
  $[x_{1:KT}]_i = f_{\text{sim}}\big(x_0, \theta_i, g(\theta_i)\big)$
  // compute posterior for all $\{\theta_i\}_{i=1}^n$ in parallel
  $\rho(\theta_i \mid O = 1) \propto \exp\{-C(\theta_i, [x_{0:KT}]_i)\}\rho(\theta_i)$
  // compute update $\phi_i \; \forall \{\theta_i\}_{i=1}^n$ in parallel
  if *phase = SVGD* **then**
    $\phi_i = \frac{1}{w} \sum_{j=1}^{n} w_j \big[\nabla_{\theta_j} k(\theta_j, \theta_i) + k(\theta_j, \theta_i) \nabla_{\theta_j} \log \rho(\theta_j \mid O) \big]$
    if *inference converged* **then**
      phase = SGD
  else if *phase = SGD* **then**
    $\phi_i = \nabla_{\theta_i} \log \rho(\theta_i \mid O = 1)$
  $\theta_i \leftarrow \theta_i + \epsilon \phi_i$ in parallel

IV. Stein Task and Motion Planning

Our method, *Stein TAMP (STAMP)*, solves for diverse task and motion plans by approaching TAMP as a joint Bayesian inference problem over discrete variables (symbolic plans) and continuous variables (geometric plans). Stein Task and Motion Planning (STAMP) begins by randomly initializing a set of candidate task and motion plans ('particles' or 'samples') and subsequently updates these via continuous and discrete SVGD. The goal of inference is to move these plans towards high density regions of a target distribution that effectively measures how optimal a plan is, based on their cost and constraints. After running inference, the plans are refined via stochastic gradient descent (SGD) to obtain exact optima in the target distribution. Formulating TAMP as a Bayesian inference problem allows us to sample optimal plans from various modes within the target distribution, which translates to finding a variety of optimal

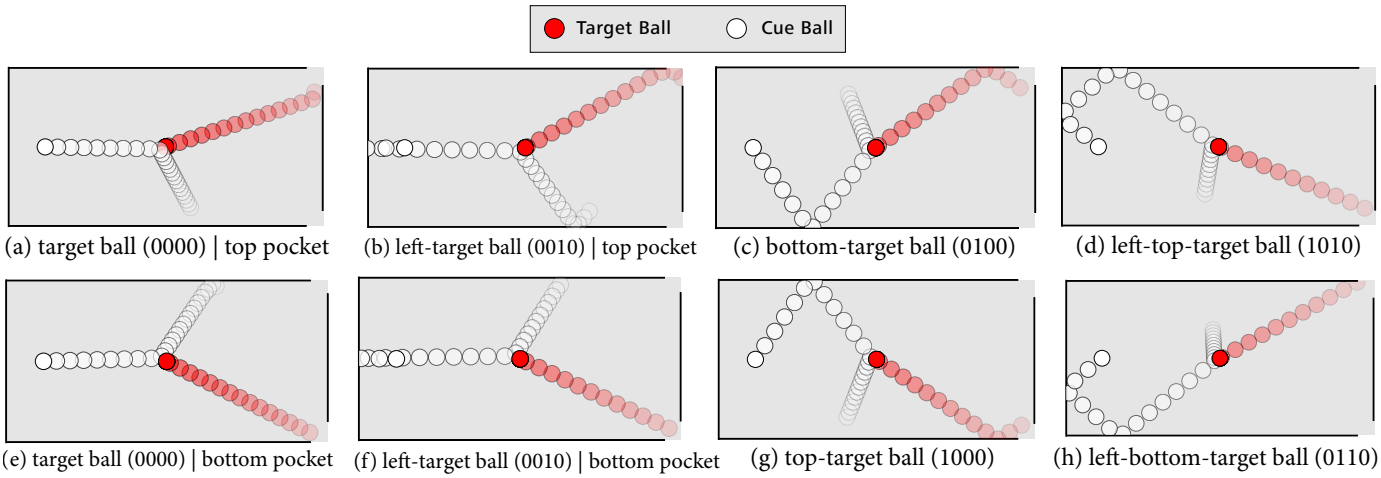


Figure 3. Sample solutions obtained from running STAMP on the billiards problem. The sub-captions indicate which walls are hit by the cue ball before hitting the target ball. That is, the first, second, third, and fourth digit correspond to hitting the top, bottom, left, and right wall, respectively. A 1 means that 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.

plans to a given TAMP problem. We illustrate our algorithm pipeline in Figure 2 and describe our approach in detail below.

A. TAMP as SVGD Inference

We first provide an analogy for STAMP in Figure 1, which illustrates using SVGD to run joint discrete-and-continuous inference on a simple Gaussian mixture. In this example, the goal is to move a set of particles towards high-density regions in the Gaussian mixture, while also ensuring that the particles obtain the correct class label post-inference. This involves inference over both discrete and continuous variables, as both the particles' class labels and their positions must be learned. In STAMP, we similarly use SVGD to run discrete-and-continuous inference. The discrete class labels are analogous to the discrete symbolic plans, and the particles' positions are analogous to the continuous geometric plans. The Gaussian mixture is analogous to a posterior distribution over the task and motion plan condition on the plan being optimal.

We now formalize our approach. Given a problem domain, we are interested in sampling optimal symbolic actions $\mathcal{A}_{1:K} \in \mathcal{Z}^K$ and controls $u_{0:KT-1} \in \mathbb{R}^{KT}$ from the posterior

$$p(\mathcal{A}_{1:K}, u_{0:KT-1} \mid O = 1)$$

where $O \in \{0, 1\}$ indicates optimality of the plan; K 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., $|\mathcal{Z}| = m$). $\mathcal{A}_{1:K}$ and $u_{0:KT-1}$ fully parameterize a task and motion plan, since they can be inputted to a physics simulator f_{sim} along with the initial state x_0 to roll out states at every timestep, that is, $x_{1:KT} = f_{\text{sim}}(x_0, \mathcal{A}_{1:K}, u_{0:KT-1})$.

We use SVGD to sample optimal plans from $p(\mathcal{A}_{1:K}, u_{0:KT-1} \mid O = 1)$. As p is defined over both continuous and discrete domains, to use SVGD's gradient-based update rule, we follow Han *et al.*'s approach of relaxing the discrete variables to the continuous domain by constructing 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 propose defining $\mathcal{Z} = \{e_i\}_{i=1}^m$[1] as a collection of m-dimensional one-hot vectors. Then, $\mathcal{A}_{1:K} \in \mathcal{Z}^K$, implying that we commit to one action (e_i) in each of the K action phases. Further, we propose using a uniform distribution for the base distribution p_0. Given this formulation, we can define a map $\Gamma : \mathbb{R}^{mK} \to \mathcal{Z}^K$ and its differentiable surrogate $\tilde{\Gamma}$ as the following (note, $x \in \mathbb{R}^{mK}$):

$$\Gamma(x) = [\max\{x_{1:m}\} \dots \max\{x_{(K-1)m+1:mK}\}]^\top \text{[2]}$$
$$\tilde{\Gamma}(x) = [\text{softmax}\{x_{1:m}\} \dots \text{softmax}\{x_{(K-1)m+1:mK}\}]^\top.$$

We show in Appendix A that the above map indeed partitions p_0 evenly when p_0 is the uniform distribution.

Our relaxation of $\mathcal{A}_{1:K}$ allows us to run inference over purely continuous variables: $a_{1:K} \in \mathbb{R}^{mK}$ and $u_{0:KT-1} \in \mathbb{R}^{KT-1}$, where $\mathcal{A}_{1:K} = \Gamma(a_{1:K})$. We denote these variables as $\theta = [a_{1:K}, u_{0:KT-1}]^\top$, and randomly initialize SVGD samples $\{\theta_i\}_{i=1}^n$ to run inference. The target distribution we aim to infer is a differentiable surrogate ρ of p, which we define following equation (3) as:

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

In practice, because SVGD requires $\nabla_\theta \log \rho$ for its updates and the gradient of $\log p_0$ is zero everywhere except at its boundary (which we can make arbitrarily large to avoid[3]), we simply let

$$\rho(a_{1:K}, u_{0:KT-1} \mid O = 1) \propto p(\tilde{\Gamma}(a_{1:K}), u_{0:KT-1} \mid O = 1). \tag{6}$$

Further, we note that it is not necessary to normalize the ρ as the normalization constant vanishes by taking the gradient of

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

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

[3] See Appendix A-A for details.

$\log \rho$.

We now quantify the surrogate of the posterior distribution. We begin by applying Bayes' rule and obtain the following factorization of $\rho(\theta \mid O = 1)$:

$$\begin{aligned} &\rho(a_{1:K}, u_{0:KT-1} \mid O = 1) \\ &\propto \rho(O = 1 \mid a_{1:K}, u_{0:KT-1})\, \rho(a_{1:K}, u_{0:KT-1}). \end{aligned} \tag{7}$$

We see that the likelihood function measures the optimality of a plan, and as such, we define it using the total cost C associated with the task and motion plan:

$$p(O{=}1|a_{1:K}, u_{0:KT-1}) \propto \exp\{-C(a_{1:K}, u_{0:KT-1}, x_{0:KT})\}. \tag{8}$$

Meanwhile, the prior $p(a_{1:K}, u_{0:KT-1})$ 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). Our formulation closely follows the mathematical program description of LGP [4], [12]–[15] which solves TAMP as a cost-minimization problem with multiple constraint functions. In our formulation, the total cost C is designed as a weighted sum of the objective function and constraints (excluding those embedded into the prior). The objective function is typically defined as a loss on the final state $x_{KT} = f_{\text{sim}}(x_0, a_{1:K}, u_{0:KT-1})$ that penalizes large deviations from the desired goal state. Constraint functions can be defined to encode the robot's kinematics and dynamics constraints, as well as constraints that enforce consistency between the trajectory and symbolic action variables.

Finally, we describe how we run inference over θ using SVGD as the inference algorithm. We first randomly initialize n candidate task and motion plans $\{\theta_i\}_{i=1}^n$, and simulate each plan in parallel using Warp [10] as the differentiable physics simulator. This allows us to generate states $x_{0:KT} = f_{\text{sim}}(x_0, \theta)$. Then, we compute the surrogate posterior distribution ρ from equations (6), (7), and (8). We use Warp's auto-differentiation capabilities to compute gradients of the log-posterior with respect to $\{\theta_i\}_{i=1}^n$ in parallel, and use these gradients to update each candidate plan θ_i via the update rule in equation (4). Since all of the above processes are parallelized on the GPU using Warp, each SVGD update is very efficient. We continue to update the plans until SVGD converges.

B. Plan Refinement via SGD

As an inference algorithm, SVGD will lead to few samples that converge exactly to optima in the posterior distribution. The repulsive force between the samples, $\nabla_\theta k(\theta, \theta')$, encourages the particles to explore, leading to diverse plans, but can simultaneously push the plans away from optima. Since we are interested in finding a diverse set of exact solutions, we exploit both SVGD and SGD in our algorithm, which allows us to balance the trade-off between exploration and optimization. After running SVGD to obtain a distribution of plans that cover multiple modes in $\rho(a_{1:K}, u_{0:KT-1} \mid O = 1)$, we finetune the candidate plans with SGD to obtain optimal solutions. This approach of combining SVGD with SGD allows us to find a diverse set of optimal plans. The pseudocode for our complete optimization routine is shown in Algorithm 1.

Table I
RUNTIME COMPARISON BETWEEN STAMP AND BASELINES

Problem	Method	Runtime[1] (s)	Factor[2]
Billiards	Diverse LGP	3509.5 ± 801.6	104 ± 3
	PDDLStream	844.0 ± 116.0	24 ± 3
	STAMP (Ours)	33.5 ± 0.4	
Pushing	Diverse LGP	2321.0 ± 425.6	13 ± 2
	PDDLStream	4615.0 ± 1639.0	30 ± 10
	STAMP (Ours)	183.9 ± 2.7	

[1] Averaged across 10 seeds; uncertainty denotes standard deviation.
[2] Indicates how much faster STAMP is on the same problem compared to the respective method; uncertainty denotes propagated uncertainty.

C. Learning Action Abstractions

Long-horizon TAMP problems are of great interest to the robotics community. One way to more effectively solve long-horizon problems is to leverage action abstractions. In our framework, we leverage *differentiable* action abstractions to reduce TAMP to a lower-dimensional inference problem. This results in more effective inference as the repulsive term in the SVGD update rule negatively correlates with the particle dimensionality [33], [34]. For STAMP, the controls $u_{0:KT-1}$ are the primary cause of high-dimensionality, particularly as the planning horizon increases (which can occur if the number of action sequences K and/or the number of timesteps T is increased).

To tackle this, we use imitation learning to create low-dimensional abstractions for $u_{0:KT-1}$, which significantly reduces the dimensionality of our inference problem. Our idea is to create a bank of goal-conditioned dynamic motion primitives (DMP) from demonstration data, whose weights we train offline. We train one DMP, or *skill,* for every possible action. Once trained, we use the DMPs to generate trajectories $\hat{x}_{kT:(k+1)T}$ given the desired goal pose g_k for each action sequence a_k. Then, we obtain the controls $u_{0:KT-1}$ using a differentiable controller that tracks $\hat{x}_{0:KT}$. This approach effectively reduces the problem from finding all of $u_{0:KT-1} \in \mathbb{R}^{KT}$ to finding $g_{1:K} \in \mathbb{R}^{Kd}$, where $d \ll T$ are the degrees of freedom of the system.

Besides dimensionality reduction, the main benefits of DMPs are that they are differentiable (a requirement of STAMP), ensure smooth trajectories $\hat{x}_{k:(k+t)T}$, and guarantee convergence to the desired goal in each segment.

V. EVALUATION

We demonstrate the effectiveness of STAMP by solving two problems requiring discrete-and-continuous planning: billiards and block-pushing. We describe these problem environments and our algorithmic setup in Section V-A below. We benchmark our algorithm against two TAMP baselines that build upon PDDLStream [3] and Diverse LGP [35], whose implementation

Table II
RUNTIME OF STAMP VARYING THE NUMBER OF PARTICLES

Particle Count	Billiards[1] (s)	Block-Pushing[1,2] (s)
100	26.1 ± 0.6	149.7 ± 0.5
200	26.3 ± 0.2	157.6 ± 0.6
400	28.1 ± 0.1	168.7 ± 0.7
800	33.5 ± 0.4	$174 \quad \pm 1$

[1] Averaged across 10 seeds; uncertainty denotes standard deviation.
[2] At most one push, i.e., $N = 1$.

Table III
RUNTIME OF STAMP VARYING THE PARTICLE DIMENSIONALITY FOR BLOCK-PUSHING

Max. Push Count	Dimensionality	Runtime[1] (s)
1	7	168.7 ± 0.7
2	16	175.5 ± 0.4
3	24	177.2 ± 0.3
4	32	176.9 ± 0.3
5	40	178.3 ± 0.5

[1] Averaged across 10 seeds; uncertainty denotes standard deviation.

we detail in Appendix E. In our experiments, we aim to answer the following questions:

(Q1) Can STAMP return a variety of accurate plans for TAMP problems for which multiple solutions are possible?
(Q2) Does STAMP conduct a more time-efficient search over symbolic and geometric plans compared to existing baselines?
(Q3) How does the computational time of TAMP scale with respect to the dimensionality of the problem and the number of samples used during inference?

A. Problem Environments and Their Algorithmic Setup

1) Billiards: We aim to find multiple ways of hitting the target ball with the cue ball such that the target ball ends up in one of the two pockets on the right side of Figure 7 in Appendix B (see Figure 3 for a variety of different solutions to this problem). This involves symbolic planning as we must plan which walls the cue ball should bounce off of before making contact with the target ball. It also involves continuous "motion planning" as we must find the initial velocity $u_0 = [v_x, v_y]^\top$ to inflict on the cue ball such that the target ball ends up in one of the pockets. We use STAMP to optimize u_0 and the task plan $\mathcal{A} = [z_1, z_2, z_3, z_4] \in \{0, 1\}^4$, which indicates which of the four walls the cue ball bounces off of. $z_i = 1$ if the cue ball bounces off of wall i, and it is 0 otherwise. We label each wall using numbers 1-4 in Figure 7 in Appendix B. In practice, we show that we can relax $\mathcal{A} \in \{0, 1\}^4$ into $a = [w_1, w_2, w_3, w_4] \in [0, 1]^4$ and express a as a function of u_0 (see Appendix B-B). Since a is a function of u_0, we simply define our SVGD samples as $\theta = [u_0]^\top$ as optimizing u_0 will implicitly optimize a. To encourage diversity in the logical plans, we employ a along with u_0 within the kernel. We refer the reader to Appendix B-C for details on how the kernel is defined for this problem.

To construct the target distribution $\rho(\theta \mid O = 1)$, we employ expressions (7) and (8). Our cost function C is a weighted sum of the target loss L_{target} and aim loss L_{aim}, with hyperparameters β_{target} and β_{aim} as the respective coefficients:

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

The target loss is minimal when the cue ball ends up in either of the two pockets, whereas the aim loss is minimal if the cue ball comes in contact with the target ball at any point in its trajectory. We define these loss terms in Appendix B-D.

2) Block-Pushing: Our goal is to push the block in Figure 8 in Appendix C towards the goal region. We wish to plan symbolic actions consisting of which side of the block—north, east, south, or west—we push against; we associate each with numbers $1, ..., 4$ as shown in Figure 8 in Appendix C. We also wish to find the forces and the block's trajectory at every time-step. For simplicity, we assume up to K action sequences can be committed *a priori*. We reduce the dimensionality of the motion planning problem by parameterizing the motion plan by a series of K intermediate goal poses $g_{x,y,\phi}^{(1:K)} \in \mathbb{R}^{3K}$, and rely on a library of DMPs, a controller, and the simulator to generate the trajectory, forces, and rolled-out states from $g_{x,y,\phi}^{(1:K)}$.

We define our samples as

$$\theta = \left[s_{1:4}^{(1)}, \ldots, s_{1:4}^{(K)}, g_{x,y,\phi}^{(1)}, \ldots, g_{x,y,\phi}^{(K)} \right]^\top. \tag{9}$$

The task plan for the kth action is $a_k = s_{1:4}^{(k)} \in \mathbb{R}^4$, which maps to a one-hot encoding e_i, $i = 1, ..., 4$ that determines which of the four sides to push on. We define the map from $\mathbb{R}^4$ to e_i to be $\Gamma\big(s_{1:4}^{(k)}\big) = \max\big\{s_{1:4}^{(k)}\big\}$. The corresponding relaxed map is $\tilde{\Gamma}\big(s_{1:4}^{(k)}\big) = \text{softmax}\big\{s_{1:4}^{(k)}\big\}$, which can be differentiated with respect to $s_{1:4}^{(k)}$ for DSVGD updates. Our base distribution $p_0(\theta)$ is the uniform distribution. We define a positive-definite kernel for θ in Appendix C-B that operates on the task and motion plans separately and accounts for angle wrap-around for $g_\phi^{(1:K)}$.

We formulate the target distribution by defining the cost as a weighted sum of the target loss L_{target} and trajectory loss $L_{\text{trajectory}}$. The target loss penalizes large distances between the cube and the goal region at the final time-step KT. The trajectory loss is a function of the intermediate goals $g_{x,y}^{(1:K)}$ and penalizes the pursuit of indirect paths to the goal. Since $L_{\text{trajectory}}$ is only a function of $g_{x,y}^{(1:K)}$, it places a prior on $g_{x,y}^{(1:K)}$, whereas L_{target} makes up the likelihood function. In summary, our target distribution is the following.

$$\begin{aligned} p(\theta \mid O = 1) &\propto \exp\{-C(\theta, x_{0:KT})\} \\ &= \exp\{-\beta_{\text{target}} L_{\text{target}}(x_{KT})\} \\ &\quad \times \exp\Big\{-\beta_{\text{trajectory}} L_{\text{trajectory}}(g_{x,y}^{(1:K)})\Big\} \end{aligned}$$

Appendix C-C describes this in greater detail.

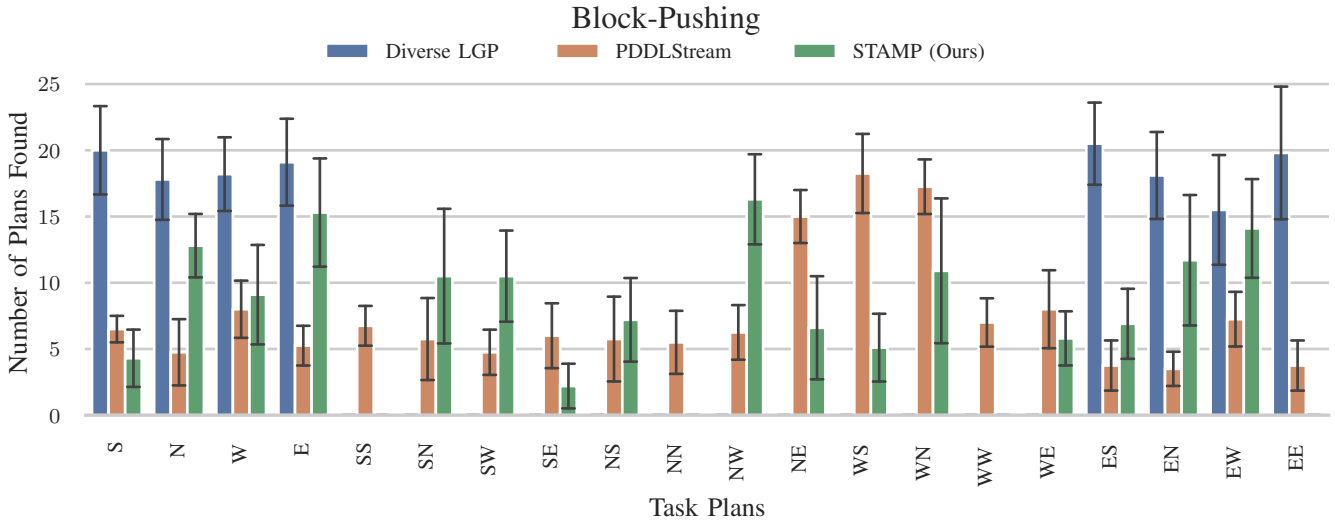


Figure 4. Distribution of plans found by STAMP (our method), Diverse LGP, and PDDLStream on the block-pushing problem (with max. two pushes). Diverse LGP and PDDLStream are invoked as many times as STAMP found solutions. Results are averaged across 10 seeds; the error bars represent the standard deviation.

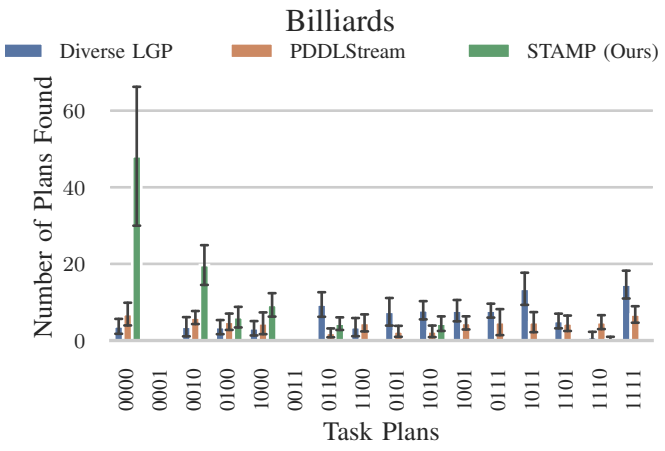


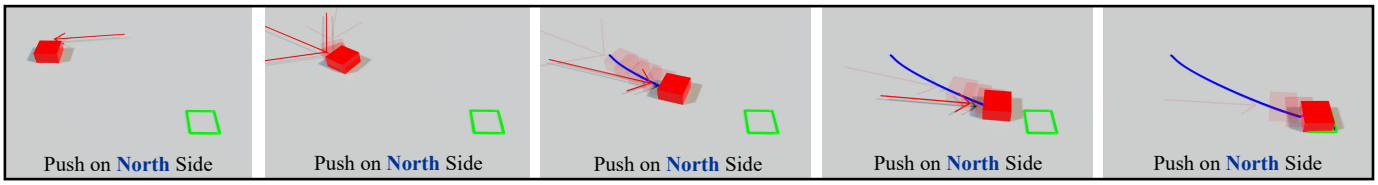
Figure 5. Distribution of plans found by STAMP (our method), Diverse LGP, and PDDLStream on the billiards problem. Diverse LGP and PDDLStream are invoked as many times as STAMP found solutions. Results are averaged across 10 seeds; the error bars represent the standard deviation.

B. Experimental Results

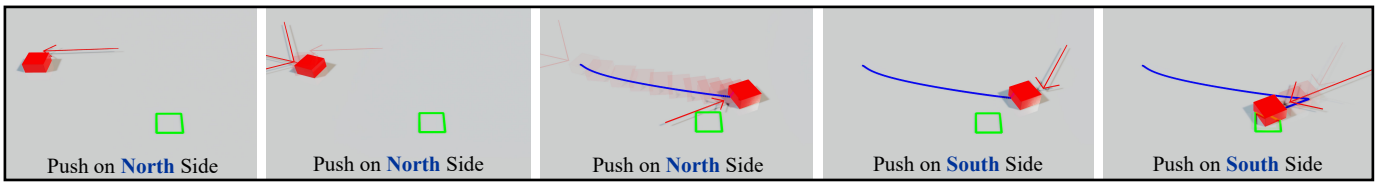
1) Solution Variety (Q1): Invoking STAMP once, we find multiple distinct plans to both problems. These plans have diverse coverage not only in terms of the types of symbolic plans found, but also the motion plans associated with each symbolic plan. STAMP's ability to generate diverse task plans is shown in Figures 4 and 5, which are histograms enumerating the number of (accurate, i.e., "good") plans found for each task plan; our results cover a broad spectrum of task plans for both problems. Figures 3 and 6 show example solutions our method found. Our experiments also show that STAMP is capable of intra-task diversity; that is, it can find multiple motion plans that accomplish the same symbolic action. For instance, in Figure 3, we see that our method found two distinct initial cue ball velocities that accomplish the task plan "0000" (i.e., a direct hit). Our observations are consistent with the intuition that the repulsive term in SVGD encourages diversity in both task and motion plans.

2) Baseline Comparison (Q2): Table I shows that STAMP is able to find the same number of solutions in substantially less time than the Diverse LGP and PDDLStream baselines for both environments. Whereas STAMP optimizes over hundreds of possible action sequences in parallel, both baselines attempt to generate a single plan in succession. Although search-based symbolic planners, like in PDDLStream, may be more efficient for determining a single solution, the ability of STAMP to reduce the time needed for motion planning becomes critical for generating multiple solutions. This distinction is apparent in the billiards problem, in which all methods use SGD to plan initial velocities but STAMP is several factors faster than the baselines. Further, STAMP learns a distribution over possible solutions, meaning the resulting modes are inherently more likely and tractable. In contrast, because diversity is forced into LGP and PDDLStream through methods like blocking or randomly selecting actions, both baselines will often consider difficult modes and be subject to the lengthy solving times needed for such modes.

3) Scalability to Higher Dimensions and Greater Number of Samples (Q3): Tables II and III show the total runtime of our algorithm while varying the number of samples and particle dimensions. They show that increasing the number of samples and the dimensions of the particles has little effect on the total optimization time required by STAMP. This is unsurprising as STAMP is parallelized over the number of samples and dimensions, made possible through the use of Warp and SVGD, both of which are parallelizable. Further, this result indicates that STAMP has the potential to efficiently solve long-horizon TAMP problems, which would require a greater number of samples and higher-dimensional inference. This is because long-horizon problems involve longer action sequences



(a) Push on north side.



(b) Push on north side, followed by push on south side.

Figure 6. Sample solutions obtained from running STAMP on the block pushing problem. The red arrow is proportional to the magnitude and direction of the pushing force, the blue line shows the block's trajectory, and the green box shows the goal region.

and naturally have highly multimodal posterior distributions due to the multitude of solutions possible, so a large number of samples would be required to run effective inference.

VI. Conclusion

We introduced STAMP, a novel algorithm for solving TAMP problems that makes use of parallelized Stein variational gradient descent and differentiable physics simulation. In our algorithm, we formulated TAMP as a variational inference problem over discrete symbolic actions and continuous motions, and validated our approach on the billiards and block-pushing TAMP problems. Our method was able to discover a diverse set of plans covering multiple different task sequences and also a variety of motion plans for the same task plan. Critically, we also showed that STAMP, through exploiting parallel gradient computation from a differentiable simulator, is much faster at finding a variety of solutions than popular TAMP solvers, and scales well to higher dimensions and more samples. In future work, we aim to test our method on more challenging TAMP problems, particularly problems that involve long-horizon planning and multiple constraints.

Acknowledgements

Y. L. is partially funded by the NSERC Canada Graduate Scholarship - Master's, Ontario Graduate Scholarship, and Google DeepMind Fellowship.

References

[1] C. R. Garrett, R. Chitnis, R. Holladay, *et al.*, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[2] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 1470–1477.

[3] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating symbolic planners and black-box samplers via optimistic adaptive planning," en, *ICAPS*, vol. 30, pp. 440–448, Jun. 2020.

[4] M. Toussaint, K. Allen, K. Smith, and J. Tenenbaum, *Differentiable physics and stable modes for Tool-Use and manipulation planning*, 2018.

[5] N. Shah, D. K. Vasudevan, K. Kumar, P. Kamojjhala, and S. Srivastava, "Anytime integrated task and motion policies for stochastic environments," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 9285–9291.

[6] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[7] 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 Computing Surveys*, 2023.

[8] Q. Liu and D. Wang, "Stein variational gradient descent: A general purpose bayesian inference algorithm," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/b3ba8f1bee1238a2f37603d90b58898d-Paper.pdf.

[9] J. Han, F. Ding, X. Liu, L. Torresani, J. Peng, and Q. Liu, "Stein variational inference for discrete distributions," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 4563–4572.

[10] M. Macklin, *Warp: A high-performance python framework for gpu simulation and graphics*, https://github.com/nvidia/warp, NVIDIA GPU Technology Conference (GTC), Mar. 2022.

[11] C. Aeronautiques, A. Howe, C. Knoblock, *et al.*, "Pddl—the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.

[12] D. Driess, J.-S. Ha, M. Toussaint, and R. Tedrake, "Learning models as functionals of Signed-Distance fields for manipulation planning," in *Proceedings of the 5th Conference on Robot Learning*, A. Faust, D. Hsu, and G. Neumann, Eds., ser. Proceedings of Machine Learning Research, vol. 164, PMLR, 2022, pp. 245–255.

[13] M. Toussaint, J.-S. Ha, and D. Driess, "Describing physics ¡italic¿for¡/italic¿ physical reasoning: Force-Based sequential manipulation planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6209–6216, Oct. 2020.

[14] M. Toussaint, "Logic-Geometric programming: An Optimization-Based approach to combined task and motion planning," en, in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, Jun. 2015.

[15] J.-S. Ha, D. Driess, and M. Toussaint, "A probabilistic framework for constrained manipulations and task and motion planning under uncertainty," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 6745–6751.

[16] J. K. Murthy, M. Macklin, F. Golemo, *et al.*, "Gradsim: Differentiable simulation for system identification and visuomotor control," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=c_E8kFWfhp0.

[17] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," *Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.

[18] Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin, "Scalable differentiable physics for learning and control," *arXiv preprint arXiv:2007.02168*, 2020.

[19] 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*, 2021.

[20] T. A. Howell, S. Le Cleac'h, J. Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A differentiable simulator for robotics," *arXiv preprint arXiv:2203.00806*, vol. 9, 2022.

[21] E. Heiden, M. Macklin, Y. Narang, D. Fox, A. Garg, and F. Ramos, "Disect: A differentiable simulation engine for autonomous robotic cutting," *arXiv preprint arXiv:2105.12244*, 2021.

[22] J. Xu, V. Makoviychuk, Y. Narang, *et al.*, "Accelerated policy learning with parallel differentiable simulation," *arXiv preprint arXiv:2204.07137*, 2022.

[23] E. Heiden, C. E. Denniston, D. Millard, F. Ramos, and G. S. Sukhatme, "Probabilistic inference of simulation parameters via parallel differentiable simulation," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 3638–3645. DOI: 10.1109/ICRA46639.2022.9812293.

[24] A. Lambert, A. Fishman, D. Fox, B. Boots, and F. Ramos, "Stein variational model predictive control," *arXiv preprint arXiv:2011.07641*, 2020.

[25] J. Envall, R. Poranne, and S. Coros, "Differentiable task assignment and motion planning,"

[26] Y. D. Zhong, J. Han, and G. O. Brikis, "Differentiable physics simulations with contacts: Do they have correct gradients wrt position, velocity and control?" *arXiv preprint arXiv:2207.05060*, 2022.

[27] 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," *arXiv preprint arXiv:2106.13281*, 2021.

[28] Y. Hu, L. Anderson, T.-M. Li, *et al.*, "Difftaichi: Differentiable programming for physical simulation," *arXiv preprint arXiv:1910.00935*, 2019.

[29] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 763–768.

[30] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, pp. 91–98, 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:14914240.

[31] H. Tan, E. Erdemir, K. Kawamura, and Q. Du, "A potential field method-based extension of the dynamic movement primitive algorithm for imitation learning with obstacle avoidance," *2011 IEEE International Conference on Mechatronics and Automation*, pp. 525–530, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:18848129.

[32] M. Ginesi, D. Meli, A. Roberti, N. Sansonetto, and P. Fiorini, "Dynamic movement primitives: Volumetric obstacle avoidance using dynamic potential functions," *Journal of Intelligent & Robotic Systems*, vol. 101, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:220280411.

[33] A. Ramdas, S. J. Reddi, B. Póczos, A. Singh, and L. Wasserman, "On the decreasing power of kernel and distance based nonparametric hypothesis tests in high dimensions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015.

[34] J. Zhuo, C. Liu, J. Shi, J. Zhu, N. Chen, and B. Zhang, "Message passing stein variational gradient descent," in *International Conference on Machine Learning*, PMLR, 2018, pp. 6018–6027.

[35] J. Ortiz-Haro, E. Karpas, M. Toussaint, and M. Katz, "Conflict-directed diverse planning for logic-geometric programming," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, no. 1, pp. 279–287, Jun. 2022. DOI: 10.1609/icaps.v32i1.19811. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/19811.

[36] D. Garreau, W. Jitkrittum, and M. Kanagawa, "Large sample analysis of the median heuristic," *arXiv preprint arXiv:1707.07269*, 2017.

[37] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, http://pybullet.org, 2019.

[38] C. R. Garrett, *Pybullet planning*, https://pypi.org/project/pybullet-planning/, 2018.

[39] C. Bäckström and B. Nebel, "Complexity results for sas+ planning," *Computational Intelligence*, vol. 11, no. 4, pp. 625–655, 1995.

[40] S. Richter and M. Westphal, "The lama planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, pp. 127–177, 2010.

[41] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, Jul. 2006. DOI: 10.1613/jair.1705.

APPENDIX A
RELAXATIONS FOR DISCRETE SVGD

Given a problem domain with m possible actions, the goal of TAMP is to find a K-length sequence of symbolic actions $\mathcal{A}_{1:K} \in Z^K$ and associated motions that solve some goal. Here, $\mathcal{Z}$ is a discrete set of m possible symbolic actions in the domain. We propose the general formulation of $\mathcal{Z}$ as the set of m-dimensional one-hot encodings such that $|\mathcal{Z}| = m$ and $|\mathcal{Z}^K| = m^K$. Then, as per equation (2), we can simply relax $\mathcal{A}_{1:K} \in Z^K$ to the real domain $\mathbb{R}^{mK}$ by constructing the map $\Gamma : \mathbb{R}^{mK} \to \mathcal{Z}^K$ as

$$\Gamma(x) = \begin{bmatrix} \max\{x_{1:m}\} \\ \max\{x_{m+1:2m}\} \\ \vdots \\ \max\{x_{(K-1)m+1:mK}\} \end{bmatrix}$$

and its differentiable surrogate as

$$\tilde{\Gamma}(\cdot) = \begin{bmatrix} \text{softmax}\{x_{1:m}\} \\ \text{softmax}\{x_{m+1:2m})\} \\ \vdots \\ \text{softmax}\{x_{(K-1)m+1:mK}\} \end{bmatrix}.$$

We will prove that the above mapping evenly partitions $\mathbb{R}^{mK}$ into m^K parts when the base distribution p_0 is a uniform distribution over $[-u, u]^{mK}$ for some $u > 0$. We will prove this in two steps: firstly by assuming $K = 1$ and then proving this for any positive integer K.

A. The Base Distribution

Our base distribution $p_0(x), x \in \mathbb{R}^{mK}$ for discrete SVGD is a uniform distribution over the domain $[-u, u]^{mK}$. That is,

$$p_0(x) = \text{unif}(x) = \begin{cases} \frac{1}{(2u)^{mK}} & \text{if } x \in [-u, u]^{mK} \\ 0 & \text{otherwise.} \end{cases}$$

Note that in practice, we can make $u \in (0, \infty)$ arbitrarily large to avoid taking gradients of p_0 at the boundaries of $[-u, u]^K$.

B. Proof of Even Partitioning: Case $K = 1$

To satisfy the condition that $\Gamma(x), x \in \mathbb{R}^m$ evenly partitions unif(x), for all $\{e_i\}_{i=1}^m \in \mathcal{Z}$, the following must be true.

$$\int_{\mathbb{R}^m} \text{unif}(x)\mathbb{I}[e_i = \Gamma(x)]\,\mathrm{d}x = \frac{1}{m}$$

Below, we show that the left hand side (LHS) of the above equation simplifies to $\frac{1}{m}$, proving that our relaxation evenly partitions $p_0(\cdot)$.

$$\begin{aligned}
\text{LHS} &= \int_{\mathbb{R}^m} \text{unif}(x)\mathbb{I}[e_i = \max(x)]\,\mathrm{d}x \\
&= \frac{1}{(2u)^m} \int_{-u}^{u} \cdots \int_{-u}^{u} \mathbb{I}[e_i = \max(x)]\,\mathrm{d}x_1 \ldots \mathrm{d}x_m \\
&= \frac{1}{(2u)^m} \int_{-u}^{u} \cdots \int_{-u}^{u} \prod_{j=i, j\neq 1}^{m} \mathbb{I}[x_i > x_j]\,\mathrm{d}x_1 \ldots \mathrm{d}x_m \\
&= \frac{1}{(2u)^m} \int_{-u}^{u} \left(\prod_{j=i, j\neq 1}^{m} \int_{-u}^{x_i} \mathbb{I}[x_i > x_j]\,\mathrm{d}x_j \right) \mathrm{d}x_i \\
&= \frac{1}{(2u)^m} \int_{-u}^{u} \left(\prod_{j=i, j\neq 1}^{m} \int_{-u}^{x_i} \mathrm{d}x_j \right) \mathrm{d}x_i \\
&= \frac{1}{(2u)^m} \int_{-u}^{u} \left(\int_{-u}^{x_i} \mathrm{d}y \right)^{m-1} \mathrm{d}x_i \\
&= \frac{1}{(2u)^m} \int_{-u}^{u} (x_i + u)^{m-1}\,\mathrm{d}x_i \\
&= \frac{1}{(2u)^m} \cdot \frac{(2u)^m}{m} \\
&= \frac{1}{m}
\end{aligned}$$

□

C. Proof of Even Partitioning: Case $K > 1$

For $\Gamma(x)$ to evenly partition unif$(x), x \in \mathbb{R}^{mK}$, the following must be true for all $\{z_i\}_{i=1}^{m^K} \in \mathcal{Z}^K$.

$$\int_{\mathbb{R}^{mK}} \text{unif}(x)\mathbb{I}[z_i = \Gamma(x)]\,\mathrm{d}x = \frac{1}{m^K}$$

We use the results from Appendices A-A and A-B to show that the left hand side of the above equation simplifies to $\frac{1}{m^K}$, proving that our relaxation evenly partitions $p_0(\cdot)$. Below, we use the following notation: $\mathbf{u} = [u \ldots u] \in \mathbb{R}^m$, $(z_i)_k$ is the kth m-dimensional one-hot vector within $z_i \in \mathcal{Z}^K$, and $\mathbf{x}_k = x_{(k-1)m+1:km} \in \mathbb{R}^m$.

$$\begin{aligned}
\text{LHS} &= \int_{\mathbb{R}^{mK}} \text{unif}(x) \prod_{k=1}^{K} \mathbb{I}[(z_i)_k = \max(\mathbf{x}_k)]\,\mathrm{d}x \\
&= \frac{1}{(2u)^{mK}} \prod_{k=1}^{K} \int_{-\mathbf{u}}^{\mathbf{u}} \mathbb{I}[(z_i)_k = \max(\mathbf{x}_k)]\,\mathrm{d}\mathbf{x}_k \\
&= \frac{1}{(2u)^{mK}} \prod_{k=1}^{K} \frac{(2u)^m}{m} \\
&= \frac{1}{(2u)^{mK}} \cdot \frac{(2u)^{mK}}{m^K} \\
&= \frac{1}{m^K}
\end{aligned}$$

□

APPENDIX B
BILLIARDS PROBLEM SETUP

A. Graphical Overview

A graphical depiction of the billiards problem environment is shown in Figure 7. We recreate this environment in the Warp simulator.

B. Task Variables as Functions of Velocity

We show here that the task variable $a = w_{1:4}$ can be formulated as a soft function of the initial cue ball velocity $u_0 = (v_x, v_y)$. To do so, we first simulate the cue ball's trajectory in a differentiable physics simulator, given the initial

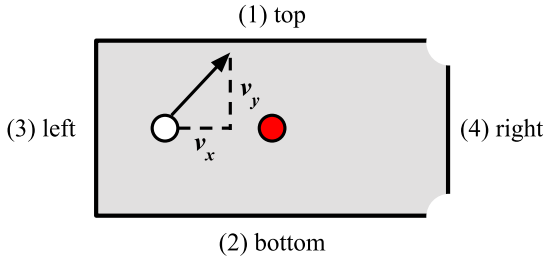


Figure 7. Graphical depiction of the billiards problem environment. The goal is to shoot the target ball (red) into one of the pockets on the right by finding an initial velocity for the cue ball (white). The task plan is give by the wall(s) the cue ball should hit before hitting the target ball.

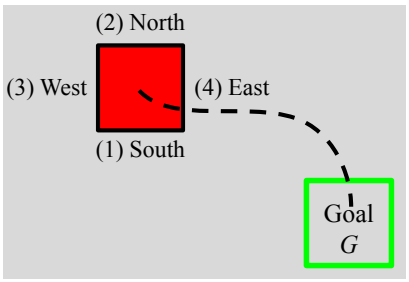


Figure 8. Graphical depiction of the block-pushing problem environment. The goal is to push the red block into the green goal region by repeatedly applying pushes to either north, south, west, or east of the red block. The task plan is given by the side(s) to push from.

cue ball velocity $u_0 = [v_x, v_y]$. We then use the rolled-out trajectory of the cue ball $x_{1:T}^{\text{cue}}$ to define a notion of distance at time t between the cue ball and wall i as follows.

$$d_t^{\text{wall-cue}} = -\alpha \text{ SignedDistance}(x_t^{\text{cue}}, \text{wall}_i) + \beta$$

In the above, $\alpha > 0, \beta \in \mathbb{R}$ are hyperparameters that can be tuned to scale and shift the resulting value.

Then, we use $d_t^{\text{wall-cue}}$ to formulate the task variable $w_{1:4}$ as a soft function of u_0 as follows.

$$w_i = \frac{1}{1 + \exp\{-d_{\text{weighted}}\}},$$
$$d_{\text{weighted}} = \sum_{t=1}^{t_c} \sigma_t d_t^{\text{wall-cue}},$$
$$\sigma_t = \frac{\exp\{d_t^{\text{wall-cue}}\}}{\sum_{k=1}^{t_c} \exp\{d_k^{\text{wall-cue}}\}}$$

Although w_i is not directly related to u_0, we note that both quantities are related implicitly since $d_t^{\text{wall-cue}}$ is a function of x_t^{cue}, which result from simulating the cube ball forward using u_0 as the initial ball velocity.

We note that the above formulation of w_i gives us the binary behavior we want for the task variable, but in a relaxed way. That is, w_i will either take on a value close to 1 or close to 0. Intuitively, the sigmoid function will assign a value close to 1 to w_i if the negative signed distance between the wall i and the cue ball is large at some point in the cue ball's trajectory up until t_c, which is the time of contact with the target ball. This can only occur if the cue ball hits wall i at some point in its trajectory before time t_c. Conversely, if the cue ball remains far from wall i at every time step up to t_c, w_i will correctly evaluate to a value close to 0. Note that α and β can be tuned to get the behavior we want for w_i.

Critically, we note that the above formulation is differentiable with respect to u_0. By relaxing w_i using a sigmoid function over d_{weighted} and using a differentiable physics simulator to roll out the cue ball's trajectory, we effectively construct w_i as soft and differentiable functions of u_0. The differentiability of w_i with respect to u_0 is important; because of the way we formulate the kernel (see Appendix B-C), the repulsive force $\nabla_\theta k(\theta, \theta')$ in the SVGD update requires gradients of w_i with respect to u_0 via the chain rule.

C. Kernel Definition

We build upon the Radial Basis Function (RBF) kernel, a popular choice in the SVGD literature, to design a positive definite kernel for $\theta = [v_x, v_y]^\top$. Whereas RBF kernels operate over θ, we operate the RBF kernel over $[\theta, w_{1:4}(\theta)]^\top$, and tune separate kernel bandwidths for $v_{x,y}$ and $w_{1:4}(v_{x,y})$, which we denote as $s_{v_{x,y}}$ and $s_{w_{1:4}}$, respectively. This is to account for differences in their range. We use the median heuristic [36] to tune the kernel bandwidths.

$$k(\theta, \theta') = \exp\left\{ -\frac{\|\theta - \theta'\|_2^2}{s_{v_{x,y}}^2} - \frac{\|w_{1:4}(\theta) - w_{1:4}(\theta')\|_2^2}{s_{w_{1:4}}^2} \right\}$$

D. Loss Definitions

We define the target and aim loss for the billiards problem below. Given the final state at time T of the target ball x_T^{target}, the position of the top pocket g_{top}, the position of the bottom pocket g_{bottom}, and the radius R of the balls, the target loss is defined as the following.

$$L_{\text{target}} = \min(\|x_T^{\text{target}} - g_{\text{top}}\|_2^2, \|x_T^{\text{target}} - g_{\text{bottom}}\|_2^2)$$

To define the aim loss, we first introduce the time-stamped aim loss $L_{\text{aim}}^{(t)}$ which is equal to the distance between the closest points on the surface of the target ball and the surface of the cue ball at time t:

$$L_{\text{aim}}^{(t)} = \max(0, \left|x_t^{\text{target}} - x_t^{\text{cue}}\right| - 2R)$$

The aim loss L_{aim} is the softmin-weighted sum of the time-stamped aim loss throughout the cue ball's time-discretized trajectory up until time t_c, which is the time of first contact with the target ball.

$$L_{\text{aim}} = \sum_{t=1}^{t_c} \sigma_t L_{\text{aim}}^{(t)}, \qquad \sigma_k = \frac{\exp\left\{-L_{\text{aim}}^{(k)}\right\}}{\sum_{t=1}^{t_c} \exp\left\{-L_{\text{aim}}^{(t)}\right\}}$$

Appendix C
Block-Pushing Problem Setup

A. Graphical Overview

Figure 8 shows a graphical overview of the block-pushing problem environment, which we recreate in the Warp simulator.

B. Kernel Definition

We employ the additive property of kernels to construct our positive-definite kernel. The kernel is a weighted sum of the RBF kernel over $\{g_{x,y}^{(k)}\}_{k=1}^{K}$, the RBF kernel over $\{\text{softmax}(s_{1:4}^{(k)})\}_{k=1}^{K}$, and the von Mises kernel over $\{g_{\phi}^{(k)}\}_{k=1}^{K}$. The von Mises kernel is a positive definite kernel designed to handle angle wrap-around.

$$\begin{aligned} K(\theta, \theta') = \sum_{k=1}^{K} \Big[& \alpha_{g_{x,y}} K_{\text{RBF}}\Big(g_{x,y}^{(k)}, g_{x,y}^{(k)'}\Big) \\ & + \alpha_{s_{1:4}} K_{\text{RBF}}\Big(\text{softmax}\big(s_{1:4}^{(k)}\big), \text{softmax}\Big(s_{1:4}^{(k)'}\Big)\Big) \\ & + \alpha_{g_{\phi}} K_{\text{VM}}\Big(g_{\phi}^{(k)}, g_{\phi}^{(k)'}\Big)\Big] \end{aligned}$$

C. Formulation of the Posterior Distribution

Our target distribution is the product of a prior $p\big(g_{x,y}^{(1:K)}\big)$ over the intermediate goals and the likelihood distribution $p(O = 1 \mid \theta)$. The prior relates to the trajectory loss $L_{\text{trajectory}}$ via the relationship $p(g_{x,y}^{(1:K)}) \propto \exp(-\beta_{\text{trajectory}} L_{\text{trajectory}})$. The trajectory loss penalizes indirect trajectories towards the goal, and we formulate it using the Manhattan distance D_{MH} and use the convention $g_{x,y}^{(0)} = x_0^{\text{cube}}$ to denote the cube's starting state.

$$p\big(g_{x,y}^{(1:K)}\big) = \prod_{k=1}^{K} p\big(g_{x,y}^{(k)} \mid g_{x,y}^{(k-1)}\big)$$

$$\begin{aligned} p\big(g_{x,y}^{(k)} \mid g_{x,y}^{(k-1)}\big) \propto \exp\Big(& -D_{\text{MH}}\big(g_{x,y}^{(k-1)}, g_{x,y}^{(k)}\big) \\ & - D_{\text{MH}}\big(g_{x,y}^{(k)}, g_{x,y}^{K}\big) \\ & + D_{\text{MH}}\big(g_{x,y}^{(k-1)}, g_{x,y}^{K}\big)\Big) \end{aligned}$$

The likelihood distribution relates to the target loss L_{target}, which measures the proximity of the cube to the goal at the end of simulation. We define the likelihood as $\exp(-\beta_{\text{target}} L_{\text{target}})$ where the target loss is defined to be the following.

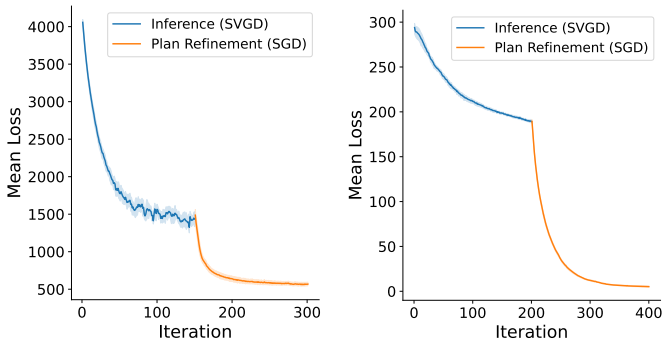
$$L_{\text{target}} = \max\big(0, \big|x_T^{\text{block}} - g_x\big| - t\big)^2 + \max\big(0, \big|x_T^{\text{block}} - g_y\big|\big)^2$$

D. Further Results

Figure 6 shows sample solutions obtained from running STAMP on the block pushing problem.

Appendix D
STAMP Optimization

Figure 9 shows how the particles move through the loss landscape of the billiards problem. Figure 10 shows how the loss evolves over various iterations of STAMP for the billiards and block-pushing problems.



(a) Billiards (b) Block-Pushing ($K = 2$)

Figure 10. Evolution of mean loss over SVGD and SGD iterations.

Appendix E
Implementation of Baselines

For both baselines, the logical or task plan is the walls to hit for the billiards problem and the sequence of sides to push from for the block pushing problem.

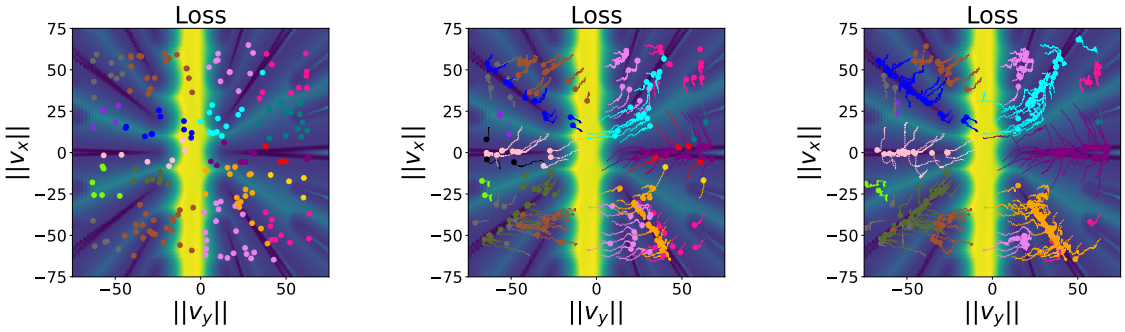
A. PDDLStream

PDDLStream combines search-based classical planners with *streams*, which construct optimistic objects for the task planner to form an optimistic plan and then conditionally sample continuous values to determine whether these optimistic objects can be satisfied [3]. Since it is possible to create an infinite amount of streams and thus optimistic objects, the number of stream evaluations required to satisfy a plan is limited and incremented iteratively. As a result, PDDLStream will always return the first-found plan with the smallest possible sequence of actions in our evaluation problems, resulting in no diversity.

To generate different solutions, we force PDDLStream to select different task plans based on some preliminary work in https://github.com/caelan/pddlstream/tree/diverse. For the billiards problem, we can reduce the task planning problem by enumerating all possible wall hits and denoting each sequence as a single task, since the motion planning stream is only needed once for any sequence of wall hits. Then, we select a task by uniform sampling and try to satisfy it by sampling initial velocities that match the desired wall hits using SGD and Warp. For the block-pushing problem, we use a top-k or diverse PDDL planner to generate multiple optimistic plans of varying complexity. We then construct the problem environment with the same tools used in PDDLStream examples [3]: first making a simulation environment in PyBullet [37], calling pose samplers and a PyBullet motion planner package [38], then attempting to track the path with a PD force controller within a stream evaluation. Note that the distribution of solutions is highly dependent on the time allocated for sampling streams. Of the feasible candidate plans, PDDLStream randomly selects a solution.

B. Logic-Geometric Programming

Diverse LGP [35] uses a two-stage optimization approach by first formulating the problem on a high, task-based, level (as an



(a) Particles at initialization (b) Particles after SVGD inference (c) Particles after SGD refinement

Figure 9. From left to right: evolution of the distribution of plan parameters θ. On the left, θ are initialized uniformly, while in the middle, they converge to the full likelihood distribution post-SVGD. On the right, the particles collapse to the optima post-SGD. The particles' colors indicate their mode (wall hits).

SAS$^+$ [39] task). Subsequently, a geometric (motion planning) problem is solved conditioned on the logical plan. That is, the performed motion is required to fulfill the logical plan. A logical plan is called *geometrically infeasible* if there is no motion plan fulfilling it. Diverse LGP speeds up exploration by eagerly forbidding geometrically infeasible plan prefixes (for instance, if moving through a wall is geometrically infeasible, no sequence starting with moving through a wall has to be tested for geometric feasibility).

We use the first iteration of LAMA [40] (implemented as part of the Fast Downward framework [41]) as suggested by Ortiz-Haro *et al.* [35] and use SGD for motion planning. To enforce diversity in the logical planner, we iteratively generate 25 logical plans by blocking ones we already found. As Diverse LGP finds exactly one solution to the TAMP problem per run, we invoke it as many times as STAMP found solutions to get the same sample size.

For both billiards and block-pushing, we use the SGD component described in Section IV-B. For billiards, we additionally add a loss term ensuring that the required walls are hit. For pusher, it suffices to fix the first $4K$ variables of (9), $s^{(1)}_{1:4}, \ldots, s^{(K)}_{1:4}$. That is, excluding them from the SGD update.