DiSProD: Differentiable Symbolic Propagation of Distributions for Planning

Palash Chatterjee, **Ashutosh Chapagain**, **Weizhe Chen** and **Roni Khardon**
Department of Computer Science, Luddy School of Informatics, Computing, and Engineering, Indiana University, Bloomington, Indiana, USA
{palchatt, aschap, chenweiz, rkhardon}@iu.edu

Abstract

The paper introduces DiSProD, an online planner developed for environments with probabilistic transitions in continuous state and action spaces. DiSProD builds a symbolic graph that captures the distribution of future trajectories, conditioned on a given policy, using independence assumptions and approximate propagation of distributions. The symbolic graph provides a differentiable representation of the policy's value, enabling efficient gradient-based optimization for long-horizon search. The propagation of approximate distributions can be seen as an aggregation of many trajectories, making it well-suited for dealing with sparse rewards and stochastic environments. An extensive experimental evaluation compares DiSProD to state-of-the-art planners in discrete-time planning and real-time control of robotic systems. The proposed method improves over existing planners in handling stochastic environments, sensitivity to search depth, sparsity of rewards, and large action spaces. Additional real-world experiments demonstrate that DiSProD can control ground vehicles and surface vessels to successfully navigate around obstacles.

1 Introduction

Planning is one of the key problems in artificial intelligence (AI), as it enables intelligent agents to make informed decisions and achieve their objectives in complex and dynamic environments. This is especially important when the environment is inherently stochastic or when the dynamics model used by the planning algorithm is imperfect. As a result, research on planning with stochastic transitions has been multifaceted, encompassing symbolic task-level planning in discrete spaces [Kolobov *et al.*, 2012; Keller and Helmert, 2013; Cui *et al.*, 2019], robotic motion planning in continuous spaces [Kurniawati *et al.*, 2011; Van Den Berg *et al.*, 2012; Agha-Mohammadi *et al.*, 2014], integrated task and motion planning [Kaelbling and Lozano-Pérez, 2013; Vega-Brown and Roy, 2018; Garrett *et al.*, 2021] and model-based reinforcement learning [Chua *et al.*, 2018; Hafner *et al.*, 2020; Curi *et al.*, 2020].

Markov Decision Processes (MDPs) provide the theoretical foundation for planning under uncertainty, but scalability remains a challenge, and approximation is often necessary. Many approaches have been proposed in the literature, including searching in action space, searching in state space, approaches using Monte-Carlo simulation, and approaches using differentiation. Several gradient-based planners have been proposed, but they are not easily usable as domain independent planners due to scalability [Deisenroth and Rasmussen, 2011], or restriction to deterministic environments [Wu *et al.*, 2020a], or in the Reinforcement Learning (RL) context, where, in most works the success of the planner depends on the quality of the learned domain-specific value function [Tamar *et al.*, 2016; Hafner *et al.*, 2020].

The paper fills this gap by introducing a novel domain-independent online planner using differentiable probabilistic transition models. Our work is related to prior algorithms for trajectory optimization, distribution propagation, and differential dynamic programming [Chua *et al.*, 2018; Williams *et al.*, 2017; Deisenroth *et al.*, 2013; Tassa *et al.*, 2012; Lenz *et al.*, 2015]. However, we introduce a novel symbolic approximation and propagation scheme generalizing work on AI planning that facilitate robustness, better approximation, and optimization [Cui *et al.*, 2019].

The main contribution of this work is in designing DiSProD (**Di**fferentiable **S**ymbolic **Pro**pagation of **D**istributions), an online planner for environments with probabilistic transitions, in continuous or hybrid spaces. The core idea is to create a symbolic graph that encodes the distribution of future trajectories conditioned on a given policy. The resulting symbolic graph provides an *analytically differentiable representation* of the policy's value, allowing efficient gradient-based optimization of the action variables for long-horizon search. While distributions over trajectories are too complex to be captured exactly, DiSProD uses Taylor's approximation and an independence assumption to facilitate a symbolic propagation of product distributions over future state and reward variables. The approximate distribution propagation can be viewed as an efficient symbolic aggregation of many trajectories, which differs from sampling algorithms that aggregate the results of many individual trajectories. This approach reduces variance in estimates in stochastic environments and facilitates planning with sparse rewards.

Extensive quantitative experiments are conducted to compare DiSProD with state-of-the-art planners in discrete-time planning in OpenAI Gym environments [Brockman *et al.*, 2016] and continuous-time control of simulated robotic systems. The results show that DiSProD outperforms existing planners in dealing with stochastic environments, sensitivity to search depth, sparsity of rewards, and large action spaces.

Furthermore, we use DiSProD with an approximate transition model to control two real-world robotic systems demonstrating that it can successfully control ground vehicles and surface vessels to navigate around obstacles.

Due to space constraints, some details are omitted from the paper. The full paper as well as code to reproduce the experiments and videos from physical experiments are available at https://pecey.github.io/DiSProD.

2 Related Work

Planning in continuous spaces has been studied in several sub-fields. A key distinction in planning methods is between *offline planners*, which compute a complete solution and then apply it, and *online planners* or Model Predictive Control (MPC) [Borrelli *et al.*, 2017], in which at every time-step, optimization is carried out over a finite horizon and only the first action from the solution is executed. The online nature of MPC provides some robustness to unexpected outcomes at the cost of increased computation time during action selection. Another important distinction is planning in *state space* (or configuration space) [LaValle, 2006] versus planning in *action space*. The former seeks an optimal sequence of states, leaving action execution to a low-level controller, while the latter produces executable actions directly. DiSProD is an *online action planner* but it can also produce a sequence of states as a byproduct, as we discuss in Section 4.6.

Within online action planners, DiSProD is related to two lines of work using differentiable transition models. First, our approach builds on the SOGBOFA algorithm [Cui *et al.*, 2018; Cui *et al.*, 2019], which was developed for discrete task-level AI planning problems. However, that work is restricted to binary state and action variables. In addition, DiSProD introduces a new distribution propagation method based on symbolic Taylor expansions. The second group includes planners using differentiable transition models in RL and control. However, deep RL work (e.g., [Heess *et al.*, 2015; Depeweg *et al.*, 2017]) makes use of learned value functions to aid planning performance and cannot plan in a new model without training first, and many approaches (e.g., [Wu *et al.*, 2020a]) use deterministic transition models. In addition, most approaches [Hafner *et al.*, 2020; Bueno *et al.*, 2019] optimize over individual trajectories and do not propagate distributions over trajectories as in DiSProD. In this realm, iLQG [Tassa *et al.*, 2012] and the PILCO family [Deisenroth *et al.*, 2013; Parmas *et al.*, 2018; Kamthe and Deisenroth, 2018] are most related to our method. iLQG linearizes the dynamics, assumes linear Gaussian transitions, as in the Extended Kalman Filter (EKF), and optimizes over individual trajectories. PILCO does propagate distributions analytically, albeit with the restricted Gaussian process (GP) dynamics and the Gaussian kernel. Gal *et al.* [2016] replace the GP in PILCO with Bayesian neural networks, which cannot propagate distributions analytically and hence requires particle-based planning. From this perspective, DiSProD can be seen as a generalization of PILCO and iLQG, which uses differentiation over approximate symbolic propagation of distribution.

Our work is also related to sampling-based planners Model Predictive Path Integral (MPPI) and Cross-Entropy Method (CEM) [Kobilarov, 2012; Williams *et al.*, 2017; Chua *et al.*, 2018; Wagener *et al.*, 2019; Mohamed *et al.*, 2022]. These algorithms sample a set of trajectories around a nominal trajectory using a fixed sampling distribution, and update the nominal trajectory based on the "goodness" of the trajectories to bias sampling in subsequent iterations. Similarly, Mania *et al.* [2018] use sampling to estimate numerical gradients which provide related policy updates. Our framework uses distribution propagation instead of sampling and it optimizes stochastic policies instead of having a fixed sampling distribution.

Finally, we note that our approach can flexibly handle both discrete and continuous variables. In contrast, other methods for planning in hybrid spaces typically restrict the transitions, for example, to piecewise linear models [Li and Littman, 2005; Zamani *et al.*, 2012; Raghavan *et al.*, 2017].

3 Algorithms and Methodology

A Markov Decision Process (MDP) is specified by $\{\mathcal{S}, \mathcal{A}, T, \mathcal{R}, \gamma\}$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, T is the transition function, $\mathcal{R}$ is the one-step reward function, and γ is the discount factor. A policy π is a mapping from states to actions. Given a policy π, the action-value function $\mathcal{Q}^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ represents the expected discounted total reward that can be obtained from state $\boldsymbol{s}_t$ if action $\boldsymbol{a}_t$ is chosen, and the policy π is followed thereafter, where $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ are vectors of state and action variables at time t. We focus on open loop probabilistic policies parameterized by $\theta = \{\theta_t\}$ where each θ_t picks the action at time step t and is further factorized over individual action variables. As in prior work, we approximate $\mathcal{Q}^{\theta}(\boldsymbol{s}_t, \theta_t) = \mathbb{E}[\sum_{i=0}^{D-1} \gamma^i \mathcal{R}(\boldsymbol{s}_{t+i}, \boldsymbol{a}_{t+i})]$ where $\boldsymbol{a}_t \sim p(\boldsymbol{a}_t|\theta_t)$ and $\boldsymbol{s}_{t+1} \sim p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, optimize θ and pick the action using θ_t. In principle the $\mathcal{Q}$-value can be calculated from the distributions over $\{(\boldsymbol{s}_{t+i}, \boldsymbol{a}_{t+i})\}$ but these distributions are complex. We approximate these distributions as products of independent distributions over individual state variables.

Our algorithm can be conceptually divided into two parts. The first calculates an approximate distribution over future states, rewards and $\mathcal{Q}^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t)$, conditioned on a given policy. A schematic overview of this process is shown in Figure 1. The second uses this representation to optimize the policy. These are described in the next two subsections.

3.1 Analytic Computation Graph

Transition Model. Typically, given $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$, a simulator T_{sim}, samples from a distribution over $\boldsymbol{s}_{t+1}$. Formally, $\boldsymbol{s}_{t+1} \sim T_{\text{sim}}(\boldsymbol{s}_t, \boldsymbol{a}_t)$. For DiSProD, we need to *encapsulate* any internal sampling in T_{sim} as an input, to separate sampling from computation, similar to the reparameterization trick.

$$\boldsymbol{s}_{t+1} = T(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\epsilon}_t) \text{ where } \boldsymbol{\epsilon}_t \sim \mathcal{N}(0, I). \qquad (1)$$

T_{sim} and its encapsulation are shown in Figure 1a and 1b.

Representation. Let $s_{t,k}$ denote the value of a random variable s_k at time t. If $s_{t,k}$ is binary, then its marginal distribution can be captured using just its mean $\hat{\mu}_{s,t,k}$ as the variance is implicitly given by $\hat{v}_{s,t,k} = \hat{\mu}_{s,t,k}(1 - \hat{\mu}_{s,t,k})$. If $s_{t,k}$ is continuous, its distribution can be summarized using its mean and variance $(\hat{\mu}_{s,t,k}, \hat{v}_{s,t,k})$. Similarly, the distribution over the noise variable ϵ_i and action variable a_ℓ are represented using $(\hat{\mu}_{\epsilon,t,i}, \hat{v}_{\epsilon,t,i})$ and $(\hat{\mu}_{a,t,\ell}, \hat{v}_{a,t,\ell})$ respectively. While the use of mean and variance suggests a normal distribution, the proposed construction does not assume anything about the form of the distribution.

Independence Assumption. To simplify the computations of propagated distributions, our approximation assumes that for all t, the distribution over the trajectories given by $p(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\epsilon}_t)$ is a product over independent factors: $p(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\epsilon}_t) = \prod_k p(s_{t,k}) \prod_\ell p(a_{t,\ell}) \prod_i p(\epsilon_{t,i})$.

Approximation. Our main observation is that the above assumption suffices to propagate approximate distributions and approximate the $\mathcal{Q}$-function. Specifically, we approximate the transition function using a second order Taylor expansion whose terms are given analytically in a symbolic form. Let $\boldsymbol{z}_t = (\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\epsilon}_t)$, $\hat{\boldsymbol{z}}_t = (\hat{\mu}_{\boldsymbol{s}_t}, \hat{\mu}_{\boldsymbol{a}_t}, \hat{\mu}_{\boldsymbol{\epsilon}_t})$ and $\boldsymbol{s}_{t+1} = T(\boldsymbol{z}_t)$. To simplify the notation, consider the j-th state variable $s_{t+1,j} = T_j(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\epsilon}_t) = T_j(\boldsymbol{z}_t)$ and let $\nabla T_j = {}^{\partial T_j}/_{\partial \boldsymbol{z}_t}$ and $H_j = {}^{\partial^2 T_j}/_{\partial \boldsymbol{z}_t \partial \boldsymbol{z}_t^\top}$. We use Taylor's expansion to approximate the encapsulated model in Equation (1) by

$$s_{t+1,j} \leftarrow T_j(\boldsymbol{z}_t) \approx T_j(\hat{\boldsymbol{z}}_t) + \nabla T_j^\top (\boldsymbol{z}_t - \hat{\boldsymbol{z}}_t) + \frac{1}{2}(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_t)^\top H_j (\boldsymbol{z}_t - \hat{\boldsymbol{z}}_t). \quad (2)$$

This is illustrated in Figure 1c. We use this approximation to calculate the mean and variance of $\boldsymbol{s}_{t+1}$ via methods of propagating distributions. Given our independence assumption, the off diagonal covariance terms multiplying H_j are zero, and the expected value of the $s_{t+1,j}$ becomes

$$\hat{\mu}_{s,t+1,j} = \mathbb{E}[s_{t+1,j}] \approx T_j(\hat{\boldsymbol{z}}_t) + \frac{1}{2}\left[\sum_k \left(\frac{\partial^2 T_j}{\partial s_{t,k}^2}\right) \hat{v}_{s,t,k} + \sum_\ell \left(\frac{\partial^2 T_j}{\partial a_{t,\ell}^2}\right) \hat{v}_{a,t,\ell} + \sum_i \left(\frac{\partial^2 T_j}{\partial \epsilon_{t,i}^2}\right) \hat{v}_{\epsilon,t,i}\right]. \quad (3)$$

A similar approximation for the variance yields 4th order moments. To reduce complexity, we use a first order Taylor approximation for the variance resulting in

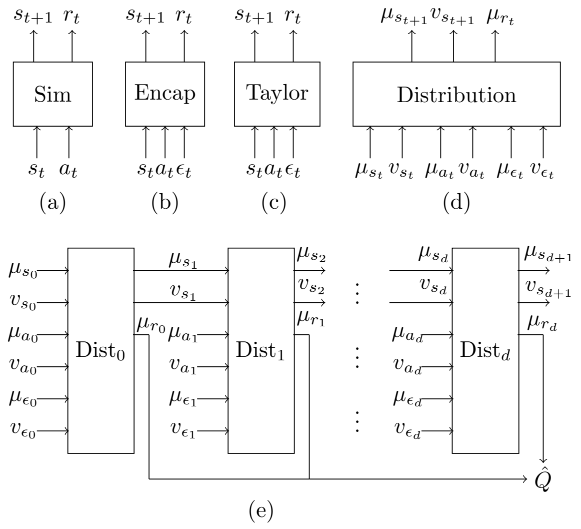
$$\hat{v}_{s,t+1,j} \approx \sum_k \left(\frac{\partial T_j}{\partial s_{t,k}}\right)^2 \hat{v}_{s,t,k} + \sum_\ell \left(\frac{\partial T_j}{\partial a_{t,\ell}}\right)^2 \hat{v}_{a,t,\ell} + \sum_i \left(\frac{\partial T_j}{\partial \epsilon_{t,i}}\right)^2 \hat{v}_{\epsilon,t,i}. \quad (4)$$

To write these concisely we collect the first order partials and diagonals of the Hessians into matrices as follows:

$$J_{\boldsymbol{s}_t}^T = \left[\frac{\partial T}{\partial \boldsymbol{s}_t^\top}\right], \qquad \text{i.e.,} \quad [J_{\boldsymbol{s}_t}^T]_{j,k} = \frac{\partial T_j}{\partial s_{t,k}}, \quad (5)$$

$$\tilde{H}_{\boldsymbol{s}_t}^T = \left[\frac{\partial^2 T}{\partial \boldsymbol{s}_t^2}^\top\right], \qquad \text{i.e.,} \quad [\tilde{H}_{\boldsymbol{s}_t}^T]_{j,k} = \frac{\partial^2 T_j}{\partial s_{t,k}^2}. \quad (6)$$

Similarly, we define $J_{\boldsymbol{a}_t}^T, \tilde{H}_{\boldsymbol{a}_t}^T, J_{\boldsymbol{\epsilon}_t}^T$, and $\tilde{H}_{\boldsymbol{\epsilon}_t}^T$ for action and noise variables, respectively. We also define $\tilde{H}_{\boldsymbol{s}_t}^{\mathcal{R}}$, $\tilde{H}_{\boldsymbol{a}_t}^{\mathcal{R}}$, and $\tilde{H}_{\boldsymbol{\epsilon}_t}^{\mathcal{R}}$ to be the second order partials of the reward function $\mathcal{R}$ with respect to the state, action and noise variables. We can now write the vector form of the equations as follows:



s_{t+1} r_t
Sim
s_t a_t
(a)
s_{t+1} r_t
Encap
$s_t a_t \epsilon_t$
(b)
s_{t+1} r_t
Taylor
$s_t a_t \epsilon_t$
(c)
$\mu_{s_{t+1}} v_{s_{t+1}}$ μ_{r_t}
Distribution
μ_{s_t} v_{s_t} μ_{a_t} v_{a_t} μ_{ϵ_t} v_{ϵ_t}
(d)
μ_{s_0} v_{s_0} μ_{a_0} v_{a_0} μ_{ϵ_0} v_{ϵ_0}
Dist$_0$
μ_{s_1} v_{s_1} μ_{r_0} μ_{a_1} v_{a_1} μ_{ϵ_1} v_{ϵ_1}
Dist$_1$
μ_{s_2} v_{s_2} μ_{r_1}
μ_{s_d} v_{s_d} μ_{a_d} v_{a_d} μ_{ϵ_d} v_{ϵ_d}
Dist$_d$
$\mu_{s_{d+1}}$ $v_{s_{d+1}}$ μ_{r_d}
$\hat{Q}$
(e)

Figure 1: Schematic overview of the idea behind the construction of analytic computation graph. (a) and (b) show the original probabilistic simulator, and its encapsulated variant where noise variables are represented as inputs. Using Taylor's approximation of this variant, we generate a third representation (c) of the same transition function. All these take as input a concrete state, action (and noise) values and compute the next state. The key idea is to combine the approximation with propagation of distributions to yield (d) that takes distributions on states, actions, and noise as input and produces a distribution over next states. Stacking this model up to the desired search depth yields symbolic propagation of distributions (e).

$$\hat{\mu}_{\boldsymbol{s}_{t+1}} \approx T(\hat{z}_t) + \frac{1}{2}\left[\tilde{H}_{\boldsymbol{s}_t}^T \hat{v}_{\boldsymbol{s}_t} + \tilde{H}_{\boldsymbol{a}_t}^T \hat{v}_{\boldsymbol{a}_t} + \tilde{H}_{\boldsymbol{\epsilon}_t}^T \hat{v}_{\boldsymbol{\epsilon}_t}\right]. \quad (7)$$

$$\hat{v}_{\boldsymbol{s}_{t+1}} \approx (J_{\boldsymbol{s}_t}^T \odot J_{\boldsymbol{s}_t}^T)\hat{v}_{\boldsymbol{s}_t} + (J_{\boldsymbol{a}_t}^T \odot J_{\boldsymbol{a}_t}^T)\hat{v}_{\boldsymbol{a}_t} + (J_{\boldsymbol{\epsilon}_t}^T \odot J_{\boldsymbol{\epsilon}_t}^T)\hat{v}_{\boldsymbol{\epsilon}_t}. \quad (8)$$

$$\hat{\mu}_{r_t} \approx \mathcal{R}(\hat{z}_t) + \frac{1}{2}\left[\tilde{H}_{\boldsymbol{s}_t}^{\mathcal{R}} \hat{v}_{\boldsymbol{s}_t} + \tilde{H}_{\boldsymbol{a}_t}^{\mathcal{R}} \hat{v}_{\boldsymbol{a}_t} + \tilde{H}_{\boldsymbol{\epsilon}_t}^{\mathcal{R}} \hat{v}_{\boldsymbol{\epsilon}_t}\right]. \quad (9)$$

In this paper, we do not model the variance of the reward function. However, this can be easily done by analogy with Equation (8) which will facilitate risk sensitive optimization, for example, conditional value at risk [Chow *et al.*, 2017]. The computations of Equations (7) to (9) are illustrated in Figure 1d. Stacking these computations over multiple time steps gives us an approximation of the distribution over future states, captured analytically as a computation graph, as shown in Figure 1e. The $\mathcal{Q}$-function is then approximated as

$$\hat{\mathcal{Q}}(\boldsymbol{s}_t, \hat{\mu}_{\boldsymbol{a}}, \hat{v}_{\boldsymbol{a}}) = \sum_{i=0}^{D-1} \gamma^i \hat{\mu}_{r_{t+i}} \quad (10)$$

where we use $\gamma = 1$ in our experiments. Note that the computation graph propagates distributions and does not sample trajectories. The computation only requires the mean $(\hat{\mu}_\epsilon = 0)$ and variance $(\hat{v}_\epsilon = 1)$ of the noise which are known in advance and are absorbed as constants in the graph.

Algorithm 1 One Step of DiSProD.

```
    Input: state
1: initialize actions (for all restarts)
2: build computation graph till depth $D$
3: while actions have not converged do
4:     loss = - $\sum_{\text{restart } k} \hat{Q}(\boldsymbol{s_t}, \hat{\mu}^k_{\boldsymbol{a}}, \hat{v}^k_{\boldsymbol{a}})$
5:     loss.backward()
6:     actions ← safe-projected-gradient-update(actions)
7: save action-means $\hat{\mu}^{k^*}_{\boldsymbol{a}_{t+1:t+D}}$ from the best restart $k^*$
8: return action $\sim \mathcal{N}(\hat{\mu}^{k^*}_{\boldsymbol{a}_t}, \hat{v}^{k^*}_{\boldsymbol{a}_t})$
```

Our construction is very general – we only require access to $T(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\epsilon}_t)$, $\mathcal{R}$ and analytic computation of their partial derivatives. In practice, T and $\mathcal{R}$ can have non-differentiable components which can be mitigated by approximating the non-smooth functions with their smoother alternatives.

3.2 Optimization Algorithm

Thanks to the symbolic representation, once the computation graph is built, it can be reused for all steps of problem solving as long as the reward function does not change. The optimization is conceptually simple, initializing all action variables for all time steps and performing gradient based search. However, some important details are discussed next.

Multiple Restarts and Loss function. Following Cui *et al.* [2019], we perform gradient search with multiple restarts. While they performed restarts sequentially, we take advantage of the structure of the computation to perform the restarts in a vectorized form. Recall that each restart is represented by $(\hat{\mu}_{\boldsymbol{a}}, \hat{v}_{\boldsymbol{a}})$ as in Equation (10). Using a superscript k to represent independent restarts we can write the loss function (negating $\mathcal{Q}$) as loss = - $\sum_{\text{restart } k} \hat{\mathcal{Q}}(\boldsymbol{s_t}, \hat{\mu}^k_{\boldsymbol{a}}, \hat{v}^k_{\boldsymbol{a}})$. Now to evaluate the loss we generate a matrix where every row represents a possible policy $(\hat{\mu}^k_{\boldsymbol{a}}, \hat{v}^k_{\boldsymbol{a}})$ and evaluate $\hat{\mathcal{Q}}$ on this matrix. Therefore, we benefit from evaluating all restarts as a "batch of examples" relative to the computation graph. Since we are optimizing with respect to the input matrix (and not computation graph weights), the loss decomposes into a sum of independent functions and gradient search effectively optimizes all restarts simultaneously.

Initialization of Actions. For discrete action variables, following Cui *et al.* [2019], we initialize actions parameters for time step $t = 0$ to binary values and for steps $t > 0$, we initialize the marginals to random values in $[0, 1]$. For continuous action variables, we must constrain actions to be in a valid interval. For each action variable a_ℓ, the policy is given by a uniform distribution, initializing $\hat{\mu}_{a,\ell} \sim \mathcal{U}(a_{\ell,\text{min}}, a_{\ell,\text{max}})$ and the variance of a uniform distribution centered around $\hat{\mu}_{a,\ell}$, i.e., $\hat{v}_{a,\ell} = {\min(a_{\ell,\text{max}} - \hat{\mu}_{a,\ell}, \hat{\mu}_{a,\ell} - a_{\ell,\text{min}})^2}/{12}$.

Policy Updates. We use Adam [Kingma and Ba, 2015] to optimize the action variables over all restarts simultaneously. We make at most $K = 10$ updates and stop the search early when $||\hat{\mu}^{\text{new}}_{\boldsymbol{a}} - \hat{\mu}^{\text{old}}_{\boldsymbol{a}}||_\infty \leq 0.1$ and $||\hat{v}^{\text{new}}_{\boldsymbol{a}} - \hat{v}^{\text{old}}_{\boldsymbol{a}}||_\infty \leq 0.01$ for normalized action ranges. Gradients are used to update both $\hat{\mu}_a$ and $\hat{v}_a$, which implies that we search over a stochastic policy. This can be important for success in sparse-reward scenarios where a stochastic policy effectively broadens the search when needed, while also allowing a nearly deterministic choice of actions at convergence.

Safe Projected-Gradient Update. The gradient-based updates have to satisfy two constraints. The first is the need to constrain variables into valid intervals. Similar to [Tassa *et al.*, 2014; Cui *et al.*, 2019], we use the standard projected gradient descent to restrict $\hat{\mu}_{\boldsymbol{a}}$ between $\boldsymbol{a}_{\text{min}}$ and $\boldsymbol{a}_{\text{max}}$, while $\hat{v}_{\boldsymbol{a}}$ is constrained to $\min({1}/{12}, {\min(\hat{\mu}_{\boldsymbol{a}} - \boldsymbol{a}_{\text{min}}, \boldsymbol{a}_{\text{max}} - \hat{\mu}_{\boldsymbol{a}})^2}/{12})$ which is the variance of the largest legal uniform distribution centered around $\hat{\mu}_{\boldsymbol{a}}$. If the gradient step pushes $\hat{\mu}_{\boldsymbol{a}}$ or $\hat{v}_{\boldsymbol{a}}$ outside the valid region, it is clipped to the boundary. Finally, we only take a gradient step on the restart if it improves the $\mathcal{Q}$-value and otherwise we maintain the previous value. This safe update requires an additional evaluation of the computation graph to check for the $\mathcal{Q}$-value improvement and it increases runtime, but it ensures that $\hat{\mathcal{Q}}$ is monotonically increasing.

Saving Actions. Since the gradients are back-propagated through the entire computation graph, action variables at all depths are adjusted according to the corresponding gradients. However, only the action at depth $d = 0$ is executed. Formally, at time t, we sample an action using $\hat{\mu}_{\boldsymbol{a}_t}$ and $\hat{v}_{\boldsymbol{a}_t}$ while $\hat{\mu}_{\boldsymbol{a}_{t+1:t+D}}$ and $\hat{v}_{\boldsymbol{a}_{t+1:t+D}}$ are not used. These updated actions can be used to initialize the action variables when planning at state s_{t+1}. The same idea has been used before in MPPI [Williams *et al.*, 2017; Wagener *et al.*, 2019]. Note that MPPI also uses multiple samples, but these samples all contribute to the update of a single action sequence, which can potentially harm the search [Lambert *et al.*, 2021; Barcelos *et al.*, 2021]. This is different in DiSProD where each restart is an independent search. To allow reuse of old search but add diversity, we initialize one restart using saved action mean, and initialize all other restarts randomly.

Overall Optimization Algorithm. These steps are summarized in Algorithm 1, where search depth (D), the initial stepsize for $\hat{\mu}_{\boldsymbol{a}}$ and $\hat{v}_{\boldsymbol{a}}$ $(lr_\mu$ and $lr_v)$, and number of restarts serve as hyper-parameters. When all restarts have converged or K gradient steps have been performed, we choose the *best restart* with the maximum $\mathcal{Q}$-value, breaking ties randomly. Finally, we sample an action using the mean and variance of the first action distribution of the best restart.

3.3 Discussion

DiSProD propagates approximate distributions over trajectories using moment matching on individual variables, assuming independence between variables. As shown by Cui *et al.* [2018], in the binary case, this is equivalent to belief propagation, which works well empirically despite lack of formal guarantees. DiSProD offers a different trade-off from sampling-based methods which are exact in the limit of infinite samples but are sensitive to variance in estimates with limited samples. In this sense, our computation graph provides a stable if biased estimate of the $\mathcal{Q}$-function. With deterministic transitions and policy, our computation is exact and gradient descent can potentially find the optimal policy. For stochastic transitions, the computation is approximate and we use a loose stopping criterion to reduce run time.

4 Experiments

We experiment with DiSProD on a variety of deterministic and stochastic environments and evaluate its robustness to long horizon planning, its ability to work with sparse rewards, and its performance in high-dimensional action spaces. For this purpose, we conduct extensive experiments on simulated and real-world environments with increasing complexity.

We use three OpenAI Gym environments (Cart Pole, Mountain Car, and Pendulum) to evaluate the *robustness* of the compared planners in terms of stochasticity, planning horizon, and reward sparsity. The original OpenAI Gym environments are deterministic. We therefore enhance these environments by explicitly adding noise into the model and using it as a part of the dynamics $T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon})$. Further details are in the full paper. For the discussion, it suffices to note that we parameterize the amount of noise using a scalar α and can therefore evaluate planners' performance as a function of stochasticity in the dynamics.

In addition, we developed a new Gym environment that models a simplified vehicle dynamics, elaborating over the Dubins' car model [Dubins, 1957]. In particular, the agent can control the change in linear (Δv_t) and angular velocity $(\Delta \omega_t)$ instead of controlling the velocities directly, and the maximum change $(\overline{\Delta v}, \overline{\Delta \omega})$ is limited to a small value. This model is still inaccurate because it ignores friction, inertia, actuation noise and localization noise, but it provides an acceptable approximation. We then use this model to plan in a physics simulator with asynchronous execution. Specifically, the planner is used to control a TurtleBot in Gazebo simulation via a Robot Operating System (ROS) interface.

Finally, to demonstrate robustness and applicability, DiSProD is used to control two physical robot systems: an Unmanned Ground Vehicle (Jackal) and an Unmanned Surface Vessel (Heron) using the aforementioned model.

In the experiments we use 200 restarts, and for DiSProD $lr_v = {}^{lr_\mu}/_{10}$. The values of D and lr_μ for Gym environments are provided in Figure 2. For TurtleBot we use $D = 100$, $lr_\mu = 10$. For Jackal and Heron, D is modified to 30, 70 respectively, and we use 400 restarts.

4.1 Baselines

We compare DiSProD to CEM and MPPI. Both are shooting-based planners that maintain a sequence of action distributions to sample actions from. From a given state, they use the sampled actions to generate multiple trajectories and compute the rewards accumulated at the end of each trajectory. The two algorithms differ in how they recompute the action distribution. While CEM uses the actions from the the top n trajectories to form the new action distribution, MPPI weights the actions in a trajectory by the cumulative reward for that trajectory. Hyperparameters for the planners were frozen after tuning them on deterministic versions of the environments.

CEM and MPPI can potentially benefit from the use of saving actions. We have found that this is helpful in the basic environments but harms their performance in the car model. To ensure a fair comparison, we use the best setting for the baselines whereas DiSProD always saves actions.

4.2 Evaluation in Basic Gym Environments

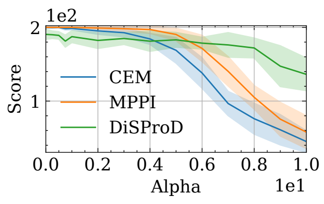
Increasing Stochasticity. First, we explore the performance of the planners with the original deterministic environments and with added stochasticity. To separate randomness in the environment from randomness in the experiment, we perform 8 repetitions over 6 runs in each environment, calculating means in each repetition and standard deviations of the means across repetitions. Averages and standard deviations are shown in Figures 2a to 2c. We observe that the planners perform similarly in deterministic environments ($\alpha = 0$) but with increasing amounts of stochasticity, DiSProD degrades more gracefully and performs better than CEM or MPPI.

Increasing Planning Horizon. Intuitively the deeper the search, the more informative it is. But for sampling-based planners, deeper search can also increase variance when estimating action quality and hence harm performance. Results for experiments testing this aspect in noisy variants of basic environments are shown in Figures 2d to 2f and deterministic environments are included in the full paper. We observe that while in the deterministic setting all the planners have similar performance, in some stochastic environments, the performance of CEM and MPPI can degrade with increasing search depth while DiSProD gives better results. In these environments, beyond a required minimum depth, increasing the depth further does not help performance. This changes, however, when we change reward sparsity.

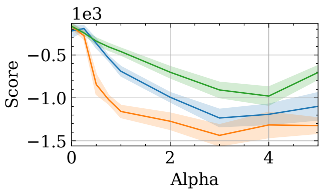
Increasing Reward Sparsity. Robust planners should be able to work with both sparse and dense rewards. Intuitively, if the reward is dense then even a short search horizon can yield good performance, but a deeper search is required for sparse rewards. To test this, we evaluate the performance of the planners by varying the sparsity of the rewards in the Mountain Car environment. With the standard reward function, the agent gets a large positive reward when its position and velocity are larger than certain thresholds. We modify the reward function to use a smooth version of greater-than-equal-to function given by $\sigma(10\beta(x - \text{target}))$ where β is a sparsity multiplier ($\beta = 1$ in earlier experiments) and $\sigma(a) = 1/(1 + e^{-a})$. The larger the value of β, the harder it is to get a reward from bad trajectories. Results are shown in Figures 2g and 2h. Figure 2g shows that with the standard search depth of 100, all planners fail to reach the goal once the reward becomes very sparse. Figure 2h shows that DiSProD can recover and perform well by increasing search depth to 200 but CEM and MPPI fail to do so.

4.3 Evaluation in High Dimensional Action Space

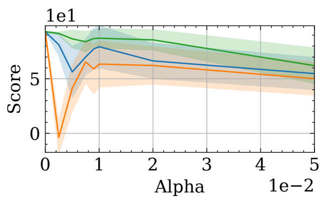
To explore high dimensional action spaces, we modify the Mountain Car environment by adding redundant action variables. The dynamics still depend on a single relevant action variable but the reward function includes additional penalties for the redundant actions. To obtain a high score, the agent must use a similar policy as before for the relevant action and keep the values of the redundant action variables as close to 0 as possible. Details of the model are in the full paper. We compare the performance of the planners against number of redundant actions, without changing any other hyperparameters. Results are shown in Figure 2i, where the MPPI results



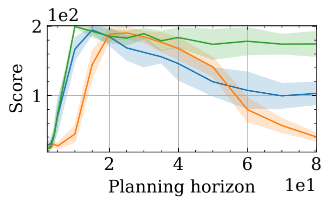
(a) CCP $(D = 25)$



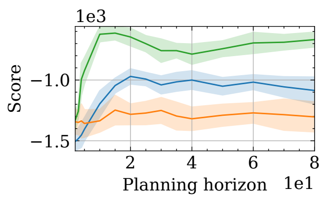
(b) P $(D = 25)$



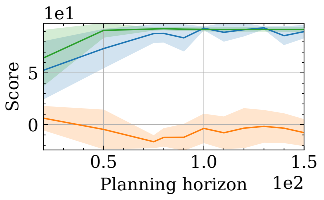
(c) CMC $(D = 100)$



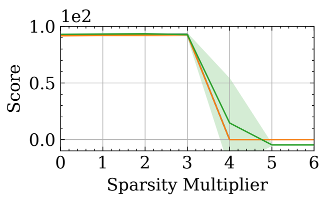
(d) CCP $(\alpha = 5)$



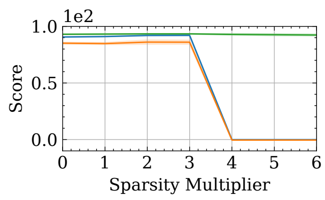
(e) P $(\alpha = 2)$



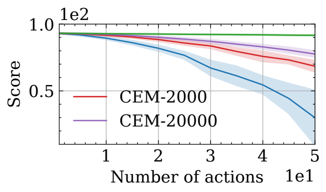
(f) CMC $(\alpha = 0.002)$



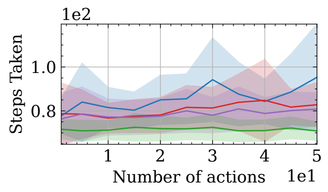
(g) CMC $(D = 100)$



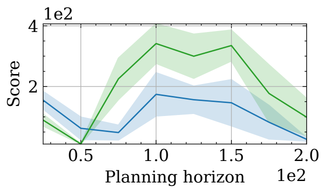
(h) CMC $(D = 200)$



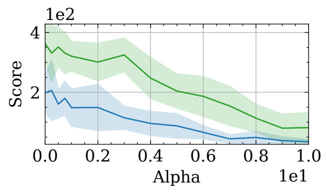
(i) CMC-HD $(D = 100, \alpha = 0)$



(j) CMC-HD $(D = 100, \alpha = 0)$



(k) CCP-Hybrid $(\alpha = 0)$



(l) CCP-Hybrid $(D = 120)$

Figure 2: Environments used are Continuous Cartpole (CCP, $lr_\mu = 10$), Pendulum (P, $lr_\mu = 1$) and Continuous Mountain Car (CMC, $lr_\mu = 0.1$). 2a, 2b, 2c: While the performance of planners is similar in deterministic environments $(\alpha = 0)$, DiSProD degrades more gracefully as compared to CEM and MPPI as the environments becomes more stochastic. 2d, 2e, 2f: In stochastic environments, using a large planning horizon can negatively impact the performance of CEM and MPPI while DiSProD gives better results. 2g, 2h: With a planning horizon of 100, all planners fail to reach the goal when the reward becomes very sparse, but DiSProD is able to recover by increasing the horizon to 200. 2i, 2j: In CMC with large action space, CEM requires 100 times the number samples to achieve comparable results to DiSProD with 200 restarts. 2k, 2l: DiSProD is able to achieve high rewards in hybrid settings as well.

are omitted since the scores are too low and they distort the plot. We observe that while CEM/MPPI perform poorly as the action space increases, the performance of DiSProD remains stable. The performance of CEM improves if we increase the population size (number of samples) by a factor of 100 (from 200 to 20,000), but it still lags behind DiSProD . Additional results analyzing this scenario are in the full paper. We note that despite the inferior reward, CEM is able to reach the goal location. However, as shown in Figure 2j, it requires a lot more steps to reach the goal. This experiment illustrates the potential advantage of planners that use the analytic model to identify what causes good outcomes as compared to estimating this effect through sampling.

4.4 Evaluation with Hybrid State Space

While we focus on experiments in continuous spaces, our planner is compatible with hybrid environments. To illustrate this we modify the Cart Pole environment to include a binary variable which is set to 1 if the cart is to the right of a certain x-coordinate. The agent receives a reward of 3 when this binary variable is set to 1, otherwise it gets a reward of 1. Figures 2k and 2l show the performance of DiSProD and CEM against planning horizon and noise level. We observe that a deeper search is required for this problem and that DiSProD can successfully obtain high reward.

4.5 Evaluation with a Physics Simulator

We control a TurtleBot using DiSProD on 16 maps with varying degrees of difficulty. In these experiments, the analytic transition model does not take obstacles into account. Incorporating obstacles in the transition yields similar or better results, but slows down the planner due to the increased size of the analytical computation graph. Instead, the reward function penalizes the agent on collision with obstacles. The obstacle patterns and detailed evaluation results are in the full paper. Following Wu *et al.* [2020b], we use success rate (SR) and success weighted by optimal path length (SL) for evaluation. SR measures the success percentage across different maps, while SL is the ratio of actual path length to the euclidean distance from starting position to goal, averaged over cases where navigation was successful. Intuitively, the lower the SL value, the faster the agent reaches the goal.

Results are shown in Table 1, averaged over 5 runs for each map and averaged over all instances. We observe that all planners are able to control the TurtleBot but DiSProD performs better in both metrics. We also evaluated the planners in a Gym environment where the planners' model matches the environment dynamics exactly. In this case, all planners perform similarly. Hence performance differences in TurtleBot are mainly due to better handling of the inaccurate model.

Environments	Method	Success Rate (SR)$\uparrow_0^{100}$	Success Length (SL)$\downarrow$
OpenAI Gym	CEM	**100.00**	**1.36**
	MPPI	97.30	1.42
	DiSProD	**100.00**	1.44
TurtleBot	CEM	85.88	1.56
	MPPI	85.88	1.62
	DiSProD	**95.29**	**1.54**

Table 1: Aggregated SR and SL for all maps when using the Dubins car model to plan in our Gym simulator and TurtleBot.

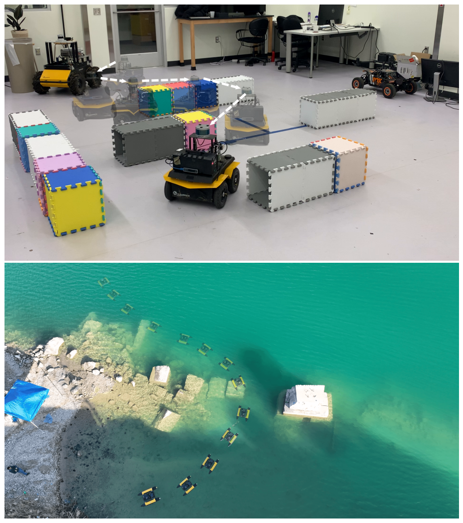


Figure 3: DiSProD controls an Unmanned Ground Vehicle and an Unmanned Surface Vessel to navigate around obstacles. The opacity indicates the robot's poses at different times.

4.6 Experiments with Robotic Systems

We use DiSProD with the same analytic model to control Jackal and Heron. Real-time control requires high frequency control commands to avoid significant drift. Therefore, one has to find a good balance between planning horizon (D) and the maximum change of linear velocity $\overline{\Delta v}$. Experiments with low $\overline{\Delta v}$ require a large D and are slow, while experiments with high $\overline{\Delta v}$ suffer from drift, as the hardware cannot stop or accelerate instantaneously. We specified these parameters through initial exploration with the systems. Other parameters are the same as in the TurtleBot simulation.

For experiments with Jackal, DiSProD is used exactly as above, i.e., sending actions to the robot. For the surface vessel, however, we use our planner in another way because our Heron has a motor issue in one of its thrusters. Specifically, DiSProD optimizes the action sequence exactly as before. Instead of sending the actions, it computes intermediate states expected to be reached with the policy (which are available in the computation graph), and sends these as "waypoints" to a PID controller. We found that, since our planner works at a fine time granularity, we can send every 5th state to a PID controller and achieve smooth control. Figure 3 visualizes the trajectories generated by DiSProD when controlling Jackal and Heron. Some videos from these experiments can be seen at https://pecey.github.io/DiSProD.

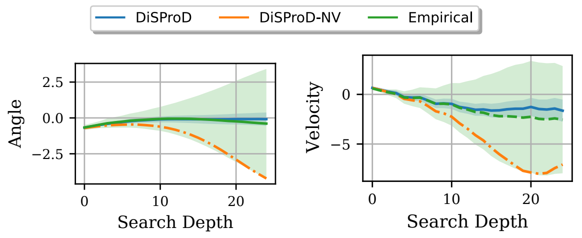


Figure 4: Comparing empirical state distributions in Pendulum ($\alpha = 1$) with approximations computed by DiSProD and DiSProD-NV.

4.7 Ablation Study and Runtime Comparison

We first evaluate the contribution of the different variance terms ($\hat{v}_{\boldsymbol{s}_t}$, $\hat{v}_{\boldsymbol{a}_t}$ and $\hat{v}_{\boldsymbol{\epsilon}_t}$) in Equations (7) and (8) to the performance of the algorithm. We use the term *complete mode* when the planner uses the variance terms (DiSProD) and *no-variance mode* when it zeroes them out (DiSProD-NV). We look into the state distributions produced by DiSProD and DiSProD-NV, and compare them against empirical state distributions. We fix the start state and a sequence of action distributions, and compute the next-state distribution for a fixed depth. For the empirical state distribution, we sample actions from the same fixed action distribution and use the dynamics model to compute the next-state. The trajectory distributions for Pendulum are visualized in Figure 4. We observe that DiSProD gives us a better approximation than DiSProD-NV – the mean is more accurate and while the variance is underestimated, it has a reasonable shape. Additional experiments in the full paper show additional visualization of trajectory distributions, as well as showing that both action and state variance contribute to the improved planning performance.

We next consider run-time comparing DiSProD to the baselines. Evaluating on the basic Gym environments (details in the full paper), DiSProD-NV has roughly the same run time as CEM and MPPI and DiSProD is up to 7 times slower. This gap is expected to increase with more state variables due to the inclusion of partial derivatives in the computation graph.

5 Conclusion

The paper presents DiSProD, a novel approach for planning in continuous stochastic environments. The method is general and overcomes limitations of prior work by using an abstracted representation, using a higher order Taylor approximation, and showing how optimization via gradients can be done over propagation of distributions. Experiments show success across multiple problems, improved handling of stochastic environments, and decreased sensitivity to search depth, reward sparsity, and large action spaces. DiSProD is also shown to be compatible with control in real robotic systems. At present, the key limitations of DiSProD are its computational complexity arising from incorporating the first and second order partials in the computation graph, over which we perform gradient search, its approximation quality requiring non-zero partial derivatives, and the need for a known model which may be resolved using model-based RL. These are left as important questions for future work.

Acknowledgements

We are grateful to members of the VAIL Lab at Indiana University, especially Durgakant Pushp and Lantao Liu, for helping with the robotic experiments. This work was partly supported by NSF under grants 2002393, 2006886, 2047169 and 2246261. Some of the experiments in this paper were run on the Big Red computing system at Indiana University, supported in part by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute.

References

[Agha-Mohammadi *et al.*, 2014] Ali-Akbar Agha-Mohammadi, Suman Chakravorty, and Nancy M Amato. Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33(2):268–304, 2014.

[Barcelos *et al.*, 2021] Lucas Barcelos, Alexander Lambert, Rafael Oliveira, Paulo Borges, Byron Boots, and Fabio Ramos. Dual Online Stein Variational Inference for Control and Dynamics. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

[Borrelli *et al.*, 2017] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Bueno *et al.*, 2019] Thiago P. Bueno, Leliane N. de Barros, Denis D. Mauá, and Scott Sanner. Deep reactive policies for planning in stochastic nonlinear domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7530–7537, 2019.

[Chow *et al.*, 2017] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *Journal of Machine Learning Research*, 18:167:1–167:51, 2017.

[Chua *et al.*, 2018] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

[Cui *et al.*, 2018] Hao Jackson Cui, Radu Marinescu, and Roni Khardon. From stochastic planning to marginal map. *Advances in Neural Information Processing Systems*, 31, 2018.

[Cui *et al.*, 2019] Hao Cui, Thomas Keller, and Roni Khardon. Stochastic planning with lifted symbolic trajectory optimization. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 119–127, 2019.

[Curi *et al.*, 2020] Sebastian Curi, Felix Berkenkamp, and Andreas Krause. Efficient model-based reinforcement learning through optimistic policy search and planning. *Advances in Neural Information Processing Systems*, 33:14156–14170, 2020.

[Deisenroth and Rasmussen, 2011] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[Deisenroth *et al.*, 2013] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423, 2013.

[Depeweg *et al.*, 2017] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. In *International Conference on Learning Representations, ICLR*, 2017.

[Dubins, 1957] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[Gal *et al.*, 2016] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, page 25, 2016.

[Garrett *et al.*, 2021] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[Hafner *et al.*, 2020] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

[Heess *et al.*, 2015] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

[Kaelbling and Lozano-Pérez, 2013] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013.

[Kamthe and Deisenroth, 2018] Sanket Kamthe and Marc Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. In *International conference on artificial intelligence and statistics*, pages 1701–1710. PMLR, 2018.

[Keller and Helmert, 2013] Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon mdps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 23, pages 135–143, 2013.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations, ICLR*, 2015.

[Kobilarov, 2012] Marin Kobilarov. Cross-entropy motion planning. *The International Journal of Robotics Research*, 31(7):855–871, 2012.

[Kolobov *et al.*, 2012] Andrey Kolobov, Peng Dai, Mausam Mausam, and Daniel Weld. Reverse iterative deepening for finite-horizon mdps with large branching factors. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, pages 146–154, 2012.

[Kurniawati *et al.*, 2011] Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 30(3):308–323, 2011.

[Lambert *et al.*, 2021] Alexander Lambert, Fabio Ramos, Byron Boots, Dieter Fox, and Adam Fishman. Stein variational model predictive control. In *Conference on Robot Learning*, pages 1278–1297. PMLR, 2021.

[LaValle, 2006] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[Lenz *et al.*, 2015] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, volume 10. Rome, Italy, 2015.

[Li and Littman, 2005] Lihong Li and Michael L. Littman. Lazy approximation for solving continuous finite-horizon mdps. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, AAAI'05, page 1175–1180. AAAI Press, 2005.

[Mania *et al.*, 2018] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[Mohamed *et al.*, 2022] Ihab S Mohamed, Kai Yin, and Lantao Liu. Autonomous navigation of agvs in unknown cluttered environments: log-mppi control strategy. *IEEE Robotics and Automation Letters*, 2022.

[Parmas *et al.*, 2018] Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. Pipps: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074. PMLR, 2018.

[Raghavan *et al.*, 2017] Aswin Raghavan, Scott Sanner, Roni Khardon, Prasad Tadepalli, and Alan Fern. Hindsight optimization for hybrid state and action mdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[Tamar *et al.*, 2016] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. *Advances in neural information processing systems*, 29, 2016.

[Tassa *et al.*, 2012] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[Tassa *et al.*, 2014] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, 2014.

[Van Den Berg *et al.*, 2012] Jur Van Den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[Vega-Brown and Roy, 2018] William Vega-Brown and Nicholas Roy. Admissible abstractions for near-optimal task and motion planning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4852–4859, 2018.

[Wagener *et al.*, 2019] Nolan Wagener, Ching an Cheng, Jacob Sacks, and Byron Boots. An online learning approach to model predictive control. In *Proceedings of Robotics: Science and Systems*, pages 1–10, 2019.

[Williams *et al.*, 2017] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

[Wu *et al.*, 2020a] Ga Wu, Buser Say, and Scott Sanner. Scalable planning with deep neural network learned transition models. *Journal of Artificial Intelligence Research*, 68:571–606, 2020.

[Wu *et al.*, 2020b] Qiaoyun Wu, Xiaoxi Gong, Kai Xu, Dinesh Manocha, Jingxuan Dong, and Jun Wang. Towards target-driven visual navigation in indoor scenes via generative imitation learning. *IEEE Robotics and Automation Letters*, 6(1):175–182, 2020.

[Zamani *et al.*, 2012] Zahra Zamani, Scott Sanner, and Cheng Fang. Symbolic dynamic programming for continuous state and action mdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 1839–1845, 2012.

Technical Appendix

A Example: Computation Graphs with Known Transition Models

For a concrete example, let us consider the dynamics of the Pendulum from the OpenAI Gym. Let $S_t = [\theta_t, \dot{\theta}_t]^\top$ be the state vector, $A_t = [a_t]^\top$ be the action vector and $\epsilon_t = [e_t]^\top$ be the noise vector. The transition function and the reward function are defined as follows:

$$T(S, A, \epsilon) = [\theta_{t+1}, \dot{\theta}_{t+1}]^\top, \tag{11}$$

$$\text{where } \dot{\theta}_{t+1} = \dot{\theta}_t + (-c_1 \sin(\theta_t + \pi) + c_2 a_t)\,\Delta_t$$

$$\theta_{t+1} = \theta_t + (\dot{\theta}_{t+1} + e_t)\Delta_t$$

$$\mathcal{R}(S, A) = -\theta_t^2 - 0.1 \times \dot{\theta}_t^{\,2} - 0.001 \times a_t^2 \tag{12}$$

$$\tag{13}$$

The original reward function uses a normalized θ_t, but we ignore it for this example, in order to keep the computations simple. Here, $c_1 = \frac{3g}{2l}$ and $c_2 = \frac{3}{ml^2}$, are constants. Then the expectation and variance of the transition function can be written as follows:

$$J^T_{S_t} = \frac{\partial S_{t+1}}{\partial S_t} = \begin{bmatrix} \frac{\partial \theta_{t+1}}{\partial \theta_t} & \frac{\partial \theta_{t+1}}{\partial \dot{\theta}_t} \\ \frac{\partial \dot{\theta}_{t+1}}{\partial \theta_t} & \frac{\partial \dot{\theta}_{t+1}}{\partial \dot{\theta}_t} \end{bmatrix}$$

$$= \begin{bmatrix} 1 - c_1 \cos(\theta_t + \pi)\Delta_t^2 & \Delta_t \\ -c_1 \cos(\theta_t + \pi)\Delta_t & 1 \end{bmatrix} \tag{14}$$

$$J^T_{A_t} = \frac{\partial S_{t+1}}{\partial A_t} = \begin{bmatrix} \frac{\partial \theta_{t+1}}{\partial a_t} \\ \frac{\partial \dot{\theta}_{t+1}}{\partial a_t} \end{bmatrix} = \begin{bmatrix} c_2 \Delta_t^2 \\ c_2 \Delta_t \end{bmatrix} \tag{15}$$

$$J^T_{\epsilon_t} = \frac{\partial S_{t+1}}{\partial \epsilon_t} = \begin{bmatrix} \frac{\partial \theta_{t+1}}{\partial e_t} \\ \frac{\partial \dot{\theta}_{t+1}}{\partial e_t} \end{bmatrix} = \begin{bmatrix} \Delta_t \\ 0 \end{bmatrix} \tag{16}$$

$$\tilde{H}^T_{S_t} = \frac{\partial^2 S_{t+1}}{\partial S_t^2} = \begin{bmatrix} \frac{\partial^2 \theta_{t+1}}{\partial \theta_t^2} & \frac{\partial^2 \theta_{t+1}}{\partial \dot{\theta}_t^2} \\ \frac{\partial^2 \dot{\theta}_{t+1}}{\partial \theta_t^2} & \frac{\partial^2 \dot{\theta}_{t+1}}{\partial \dot{\theta}_t^2} \end{bmatrix}$$

$$= \begin{bmatrix} c_1 \sin(\theta_t + \pi)\Delta_t^2 & 0 \\ c_1 \sin(\theta_t + \pi)\Delta_t & 0 \end{bmatrix} \tag{17}$$

$$\tilde{H}^T_{A_t} = \frac{\partial^2 S_{t+1}}{\partial A_t^2} = \begin{bmatrix} \frac{\partial^2 \theta_{t+1}}{\partial a_t^2} \\ \frac{\partial^2 \dot{\theta}_{t+1}}{\partial a_t^2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{18}$$

$$\tilde{H}^T_{\epsilon_t} = \frac{\partial^2 S_{t+1}}{\partial \epsilon_t^2} = \begin{bmatrix} \frac{\partial^2 \theta_{t+1}}{\partial e_t^2} \\ \frac{\partial^2 \dot{\theta}_{t+1}}{\partial e_t^2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{19}$$

$$\hat{\boldsymbol{v}}_{S_t} = \begin{bmatrix} \hat{v}_{\theta,t} \\ \hat{v}_{\dot{\theta},t} \end{bmatrix} \qquad \hat{\boldsymbol{v}}_{A_t} = [\hat{v}_{a,t}] \qquad \hat{\boldsymbol{v}}_{\epsilon_t} = [\hat{v}_{e,t}] \tag{20}$$

$$\mathbb{E}[T(S_t, A_t, \epsilon)] \approx T(\mu_{S_t}, \mu_{A_t}, \mu_\epsilon) + 0.5\left(\tilde{H}^T_{S_t}\hat{\boldsymbol{v}}_{S_t} + \tilde{H}^T_{A_t}\hat{\boldsymbol{v}}_{A_t} + \tilde{H}^T_{\epsilon_t}\hat{\boldsymbol{v}}_{\epsilon_t}\right) \tag{21}$$

$$\mathbb{V}[T(S_t, A_t, \epsilon)] \approx (J^T_{S_t} \odot J^T_{S_t})\hat{\boldsymbol{v}}_{S_t} + (J^T_{A_t} \odot J^T_{A_t})\hat{\boldsymbol{v}}_{A_t} + (J^T_{\epsilon_t} \odot J^T_{\epsilon_t})\hat{\boldsymbol{v}}_{\epsilon_t} \tag{22}$$

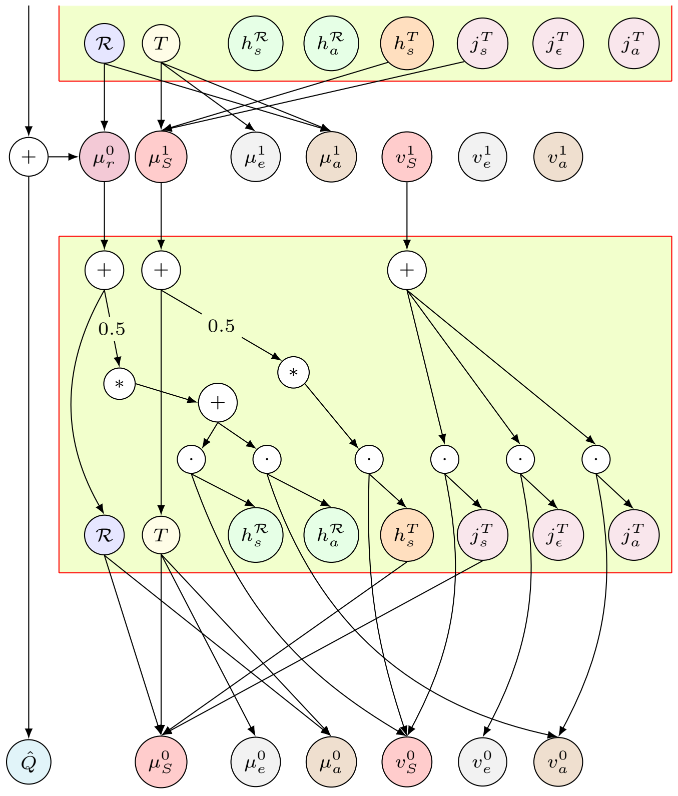


Figure 5: Computational diagram of the proposed planner for Pendulum showing the distribution block for depth 1 and a portion of the block for depth 2, as well as showing how the reward is accumulated in $\hat{Q}$. Node T and $\mathcal{R}$ represents the transition and reward function respectively. $\mu_S^0 = S_0$ and $v_S^0 = 0$ capture the initial state distribution. μ_a^i and v_a^i capture the action distribution at depth i. Similarly $\mu_e^j = 0$ and $v_e^j = 1$ capture the error distribution at depth j. Nodes h_s^T and $j_s^T, j_\epsilon^T, j_a^T$ help in computing the next state mean and variance and node $h_s^{\mathcal{R}}$ and $h_a^{\mathcal{R}}$ help in computing the mean reward. Note that h_a^T, h_ϵ^T are both zero and hence they are omitted from the graph. The nodes $h_s^{\mathcal{R}}$, $h_a^{\mathcal{R}}$, j_ϵ^T and j_a^T have a constant value and therefore they do not have incoming edges.

Similarly, the expressions for the reward function are:

$$\tilde{H}^R_{S_t} = \begin{bmatrix} \frac{\partial^2 \mathcal{R}}{\partial \theta_t^2} & \frac{\partial^2 \mathcal{R}}{\partial \dot{\theta}_t^2} \end{bmatrix} = [-2 \quad -0.2] \tag{23}$$

$$\tilde{H}^R_{A_t} = \begin{bmatrix} \frac{\partial^2 \mathcal{R}}{\partial a_t^2} \end{bmatrix} = [-0.002] \tag{24}$$

$$\mathbb{E}[\mathcal{R}(S_t, A_t)] \approx \mathcal{R}(\mu_{S_t}, \mu_{A_t}) + 0.5\left(\tilde{H}^R_{S_t}\hat{\boldsymbol{v}}_{S_t} + \tilde{H}^R_{A_t}\hat{\boldsymbol{v}}_{A_t}\right) \tag{25}$$

This illustrates that we can have analytic gradients for the model as mentioned above. Note that, we do not model the variance of the reward function as mentioned in Section 3.1. For the sake of simplicity, we have considered just a single noise variable in this model. However, the same approach works with multiple noise variables.

The computation graph for this example is shown in Figure 5. To make the graph simple, certain portions have been abstracted away. More specifically, T signifies the transition function and $\mathcal{R}$ specifies the reward function. For this example, nodes h_s^T, j_s^T, j_a^T and j_ϵ^T are used to compute the expectation and variance of next state variables and are defined as follows: $h_s^T = \tilde{H}_{S_t}^T$; $j_s^T = J_{S_t}^T \odot J_{S_t}^T$; $j_a^T = J_{A_t}^T \odot J_{A_t}^T$; $j_\epsilon^T = J_{\epsilon_t}^T \odot J_{\epsilon_t}^T$. Similarly, nodes $h_s^{\mathcal{R}}$ and $h_a^{\mathcal{R}}$ are used to compute the expected reward and are defined as: $h_s^{\mathcal{R}} = \tilde{H}_{S_t}^{\mathcal{R}}$; $h_a^{\mathcal{R}} = \tilde{H}_{A_t}^{\mathcal{R}}$.

B Contribution of Sub-Expressions in the Taylor's Expansion

As described in Section 3.1, the vector form of the equations are as follows:

$$\hat{\mu}_{\boldsymbol{s}_{t+1}} \approx T(\hat{\boldsymbol{z}}_t) + 0.5\left[\tilde{H}_{\boldsymbol{s}_t}\hat{v}_{\boldsymbol{s}_t} + \tilde{H}_{\boldsymbol{a}_t}\hat{v}_{\boldsymbol{a}_t} + \tilde{H}_{\boldsymbol{\epsilon}_t}\hat{v}_{\boldsymbol{\epsilon}_t}\right] \quad (26)$$

$$\hat{v}_{\boldsymbol{s}_{t+1}} \approx (J_{\boldsymbol{s}_t} \odot J_{\boldsymbol{s}_t})\hat{v}_{\boldsymbol{s}_t} + (J_{\boldsymbol{a}_t} \odot J_{\boldsymbol{a}_t})\hat{v}_{\boldsymbol{a}_t} + (J_{\boldsymbol{\epsilon}_t} \odot J_{\boldsymbol{\epsilon}_t})\hat{v}_{\boldsymbol{\epsilon}_t} \quad (27)$$

where $\hat{\boldsymbol{z}}_t = (\hat{\mu}_{\boldsymbol{s}_t}, \mu_{\boldsymbol{a}_t}, \mu_{\boldsymbol{\epsilon}_t})$

The equations depend on the distribution of state, action and noise variables. To analyze the impact each set of variables has on the outcome, we consider three cases - no-variance, state-variance and complete.

No variance. In this setup, all the variance terms($\hat{v}_{\boldsymbol{s}}, \hat{v}_{\boldsymbol{a}}, \hat{v}_{\boldsymbol{\epsilon}}$) are zeroed out. So the equations become

$$\hat{\mu}_{\boldsymbol{s}_{t+1}} \approx T_j(\hat{\boldsymbol{z}}) \quad (28)$$

$$\hat{v}_{\boldsymbol{s}_{t+1}} \approx 0 \quad (29)$$

State variance. Now, we just zero out action variance ($\hat{v}_{\boldsymbol{a}}$), which means we effectively optimize a deterministic policy.

$$\hat{\mu}_{\boldsymbol{s}_{t+1}} \approx T(\hat{\boldsymbol{z}}_t) + 0.5\left[\tilde{H}_{\boldsymbol{s}_t}\hat{v}_{\boldsymbol{s}_t} + \tilde{H}_{\boldsymbol{\epsilon}_t}\hat{v}_{\boldsymbol{\epsilon}_t}\right] \quad (30)$$

$$\hat{v}_{\boldsymbol{s}_{t+1}} \approx (J_{\boldsymbol{s}_t} \odot J_{\boldsymbol{s}_t})\hat{v}_{\boldsymbol{s}_t} + (J_{\boldsymbol{\epsilon}_t} \odot J_{\boldsymbol{\epsilon}_t})\hat{v}_{\boldsymbol{\epsilon}_t} \quad (31)$$

Note that at $t = 0$, as the planner knows the state exactly, therefore $\hat{v}_{\boldsymbol{s}_0} = 0$. We now consider the possible values the remaining partials might take and analyze the impact.

1. $J_{\boldsymbol{\epsilon}_t} = 0$ and $\tilde{H}_{\boldsymbol{\epsilon}_t} = 0$. Then,
$$\hat{\mu}_{\boldsymbol{s}_1} \approx T(\hat{z}_0) \quad \text{and} \quad \hat{v}_{\boldsymbol{s}_1} \approx 0 \quad (32)$$
As $\hat{v}_{\boldsymbol{s}_t}$ always remains zero, the result is similar to the no-variance mode.
2. $J_{\boldsymbol{\epsilon}_t} \neq 0$ but $\tilde{H}_{\boldsymbol{\epsilon}_t} = 0$. Then,
$$\hat{\mu}_{\boldsymbol{s}_1} \approx T(\hat{z}_0) \quad \text{and} \quad \hat{v}_{\boldsymbol{s}_1} \approx (J_{\boldsymbol{\epsilon}_t} \odot J_{\boldsymbol{\epsilon}_t})\hat{v}_{\boldsymbol{\epsilon}_t} \quad (33)$$
As $\hat{v}_{\boldsymbol{s}_1} \neq 0$, it influences $\hat{\mu}_{\boldsymbol{s}_2}$ and $\hat{v}_{\boldsymbol{s}_2}$ and future timesteps.
3. $J_{\boldsymbol{\epsilon}_t} = 0$ but $\tilde{H}_{\boldsymbol{\epsilon}_t} \neq 0$. This can happen as we evaluate the partials at the mean of the variables, and we assume zero-mean noise ($\hat{\mu}_{\boldsymbol{\epsilon}_t} = 0$). So while $J_{\boldsymbol{\epsilon}_t}$ exists, it can evaluate to zero. In such a scenario, we have,
$$\hat{\mu}_{\boldsymbol{s}_1} \approx T(\hat{z}_0) + \tilde{H}_{\boldsymbol{\epsilon}_t}\hat{v}_{\boldsymbol{\epsilon}_t} \quad \text{and} \quad \hat{v}_{\boldsymbol{s}_1} = 0 \quad (34)$$
As $\hat{v}_{\boldsymbol{s}_{t+1}} \approx (J_{\boldsymbol{s}_t} \odot J_{\boldsymbol{s}_t})\hat{v}_{\boldsymbol{s}_t}$ and $\hat{v}_{\boldsymbol{s}_0} = 0$, the next-state variance is always zero. We can generalize and write
$$\hat{\mu}_{\boldsymbol{s}_{t+1}} \approx T(\hat{z}_t) + \tilde{H}_{\boldsymbol{\epsilon}_t}\hat{v}_{\boldsymbol{\epsilon}_t} \quad \text{and} \quad \hat{v}_{\boldsymbol{s}_{t+1}} = 0 \quad (35)$$

Complete. This is the default variant of DiSProD which uses all the variance terms. As discussed above, for successful propagation of distribution, DiSProD requires at least some of the J or $\tilde{H}$ terms to be non-zero.

C Experiment details

C.1 Environments

In this section, we briefly describe the environments used for our experiments. The basic environments are taken from OpenAI Gym to which we make minor modifications. We rewrite the transition and reward functions by replacing any step-functions with their smooth equivalents. As the environments are deterministic, we add noise explicitly to make them stochastic. ϵ denotes noise sampled from a standard Gaussian and α controls the amount of noise being added.

Cartpole

Cartpole has four state variables - $x, y, \theta, \dot{\theta}$ and one control variable which controls the force acting on the cart. We make the environment stochastic by adding Gaussian noise to the force being applied. Concretely, the updated equation for force becomes $\text{force}_{\text{noisy}} = \text{force} + \alpha\epsilon$.

Mountain Car

Mountain Car has two state variables - position (x) and velocity (v) and one control variable (u) which is the force to be applied on the car. At a high level, the environment has two functions f_x and f_v that are used to update the position and velocity.

$$v_{t+1} = f_v(x_t, v_t) \text{ and } x_{t+1} = f_x(x_t, v_{t+1})$$

We add noise to v_{t+1} to make the environment stochastic. $v_{t+1} = f_v(x_t, v_t) + \alpha\epsilon$.

Pendulum

Pendulum also has two state variables - θ and $\dot{\theta}$ and the control variable (u) is the torque. First, $\dot{\theta}_{t+1}$ is computed using θ_t, $\dot{\theta}_t$, u_t and some constants, and then θ_{t+1} is updated using $\dot{\theta}_{t+1}$. We augment this and add noise to the update equation for θ_{t+1} such that it becomes $\theta_{t+1} = \theta_t + (\dot{\theta}_{t+1} + \alpha\exp(\epsilon))dt$.

Dubins Car

The standard Dubins Car model maintains three state variables - x, y, θ and has two control variables - velocity (v) and angular velocity (ω). The dynamics equations are:

$$x_{t+1} = x_t + v_t \cos\theta_t dt \quad (36)$$

$$y_{t+1} = y_t + v_t \sin\theta_t dt \quad (37)$$

$$\theta_{t+1} = \theta_t + \omega dt \quad (38)$$

Since this is a discrete time system and ignores acceleration, the motion of the car can be unrealistic. For example, although $v_t = 0$ and $v_{t+1} = v_{\max}$ is a valid control, this cannot be performed by a real vehicle. As Gazebo is a real-physics simulator and considers factors like acceleration, inertia and friction, planning with this model leads to poor performance in Gazebo. In order to obtain a better approximation of the model used by Gazebo, we modify the controls from velocity

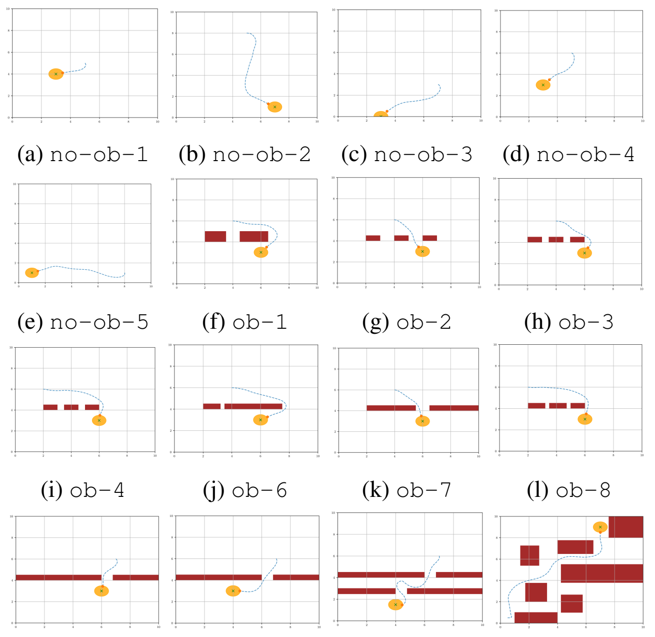

(a) `no-ob-1` (b) `no-ob-2` (c) `no-ob-3` (d) `no-ob-4`
(e) `no-ob-5` (f) `ob-1` (g) `ob-2` (h) `ob-3`
(i) `ob-4` (j) `ob-6` (k) `ob-7` (l) `ob-8`
(m) `ob-9` (n) `ob-10` (o) `ob-11` (p) `cave-mini`

Figure 6: Examples of maps on which experiments with Dubins' car in Gym were performed. Corresponding maps were generated for experiments in Gazebo.

and angular velocity to delta velocity (Δv) and delta angular velocity ($\Delta\omega$) and compute velocity and angular velocity as

$$v_t = v_{t-1} + \Delta v_t \tag{39}$$
$$\omega_t = \omega_{t-1} + \Delta\omega_t \tag{40}$$

This prevents sudden changes in linear and angular velocity and helps to limit the actions. We experiment with this model on a suite of maps ranging from simple ones with no obstacles to complex ones with obstacles. All the map configurations are shown in Figure 6 and two of the RViz maps are shown in Figure 7.

Simple Env

In the Gym environments that we experimented with, some of the required partial derivatives were either zero or not significant enough to make a difference in the planning results. To showcase the difference more clearly, we designed an additional environment called SimpleEnv where all the required partials are non-zero.

The environment has two state variables - x and y and two action variables - δ_x and δ_y. The dynamics are as follows:

$$x_{t+1} = x + \delta_x + \alpha(0.1\epsilon + \epsilon^2) \tag{41}$$
$$y_{t+1} = y + \delta_y \tag{42}$$

The reward function is a smoothed 0-1 reward where a reward of 1 is given if the agent is sufficiently close to a predefined goal position.

Cartpole - Hybrid Variant

To experiment with hybrid environments, we modify Cartpole and introduce an environment variable called *reward marker* (set to 0.5 in the experiment) and a new binary state variable that is set to 1 if the cart is to the left of the reward marker. The reward function is modified such that the agent gets a reward of 3 if the agent is to right of the reward marker and otherwise it continues to get a reward of 1. The initial position of the cart is always to the left of the reward marker. The noise is added in a similar manner as in the original Cartpole environment.

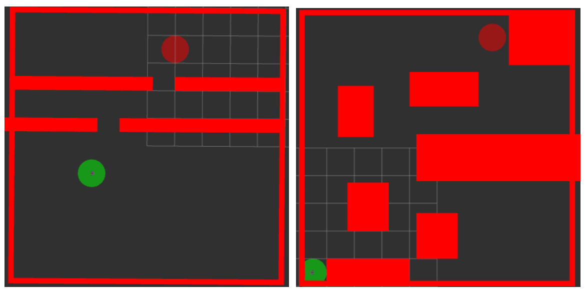

(a) Map `ob-11` in RViz (b) Map `cave-mini` in RViz

Figure 7: Two of the maps used in simulated Turtelbot experiments in Gazebo, as visualized by RViz.

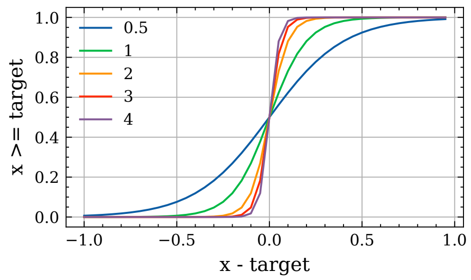


Figure 8: As the value of sparsity multiplier increases, the greater-than-equal-to grows closer to a step function.

Mountain Car - High Dimensional Action Space

For our experiments with high dimensional action space, we tweak the standard Mountain Car environment and add m redundant action variables. Similar to the original Mountain Car environment, the dynamics are only influenced by the 1st action variable. We modify the action penalty term in the reward function to sum over all the action variables. Specifically, the modified action penalty term function becomes $0.1\sum_{i=1}^{m} a_i$.

C.2 Controlling Sparsity of Rewards

As mentioned in Section 4.2, we modify the default reward function of Continuous Mountain Car to use a smooth version of greater-than-equal-to function (f_{ge}). The function takes as input a variable x and a target t and outputs $\sigma(10\beta(x-t))$ where β is the sparsity multiplier and $\sigma(a) = {}^{1}/_{1+e^{-a}}$. The function is a smooth approximation of $x \geq t$. Figure 8 shows a plot of f_{ge} against $x-t$. As the sparsity multiplier increases, the value of the function becomes increasingly similar to that of a step function and rewards become sparse.

C.3 Hardware Platforms

We use two physical robots in our experiments - Jackal and Heron, both of which are made by Clearpath Robotics. As mentioned in Section 4.5, in both the cases, the positions of obstacles in the environment are unknown to the transition model. However, the reward function is aware of the obstacles and penalizes trajectories that collide with one.

Jackal is an Unmanned Ground Vehicle (UGV), whose position and orientation are obtained through a VICON Motion Capture System, and the UGV is controlled directly by DiSProD through commands for linear and angular velocity.

Heron is an Unmanned Surface Vessel (USV), whose position is obtained by fusing GPS and Inertial Measurement Unit (IMU) information using EKF. It is also controlled through commands for linear and angular velocity, but in our experiments this is mediated through a PID controller because one of the USV's thrusters has intermittent motor failures, which requires a high-frequency feedback controller to compensate the significant error. DiSProD computes an action sequence through its computation graph and then send a list of waypoints – the means of states on the graph – to the PID controller. The list is updated asynchronously at every decision cycle of the planner.

C.4 Additional Details

We list the values of the parameters used for our experiments with OpenAI Gym in Table 2 and for our experiments with Jackal and Heron in Table 3. The step sizes for $\mu_{\boldsymbol{a}}$ and $v_{\boldsymbol{a}}$ are indicated by lr_μ and lr_v. $\overline{\Delta v}$ and $\overline{\Delta \omega}$ represent the maximum permissible values for delta linear velocity and delta angular velocity, while $\overline{v}$ and $\overline{\omega}$ indicate the maximum permissible linear and angular velocity respectively.

Environments	nA	nS	lr_μ	lr_v	Rollout Depth	Restarts
CartPole	1	4	10	1	25	200
Pendulum	1	4	1	0.1	25	200
Mountain Car	1	2	0.1	0.001	100	200
Dubins Car	5	5	10	1	100	200
Simple Env	2	2	0.01	0.001	20	50

Table 2: Parameter specifications for experiments with OpenAI Gym simulators.

Robots	$\overline{\Delta v}$	$\overline{\Delta \omega}$	$\overline{v}$	$\overline{\omega}$	Rollout Depth	Restarts
TurtleBot	0.05	1	0.5	60	100	200
Jackal	0.3	5	0.6	60	30	400
Heron	0.5	10	1	30	70	400

Table 3: This table elaborates the parameters of the Dubin's car model with action ranges, and specifies modifications to the depth and restarts in experiments with TurtleBot, Jackal and Heron.

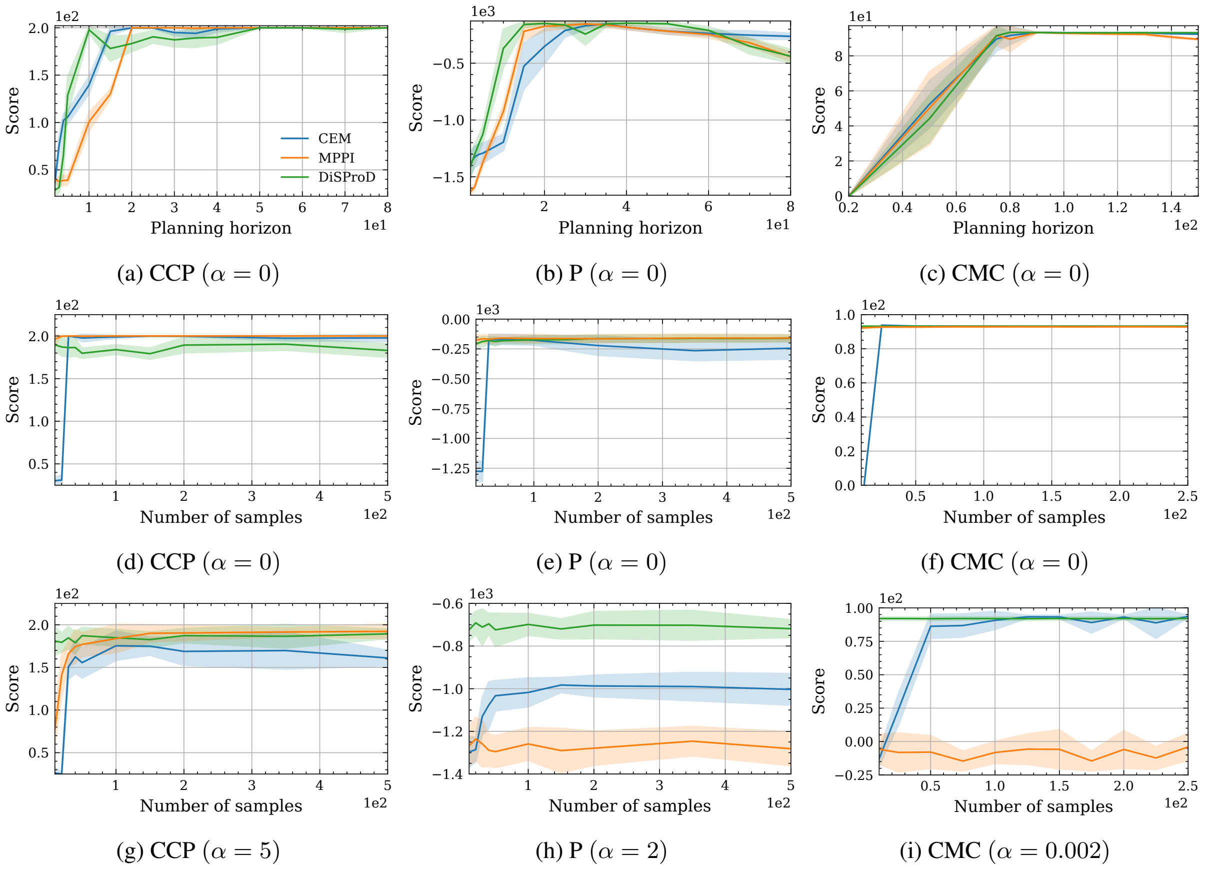
D Experimental Results

Does having a larger planning horizon help in deterministic environments? In our experiments with basic Gym environments, we observe that optimal performance, with all the three planners, is achieved with a small planning horizon. Increasing the planning horizon further does not impact the performance. (Figure 9 (a-c)).

Does sampling more trajectories help? Shooting algorithms sample action sequences to generate trajectories. Intuitively, increasing the number of action sequences that a planner is allowed to sample should help with better plans and hence improve performance. We observe that in deterministic environments, sampling a small number of trajectories is sufficient to obtain near optimal performance. Making the environments stochastic degrades the planners' overall performance. Interestingly enough, in these environments, the performance remains unaffected on increasing the number of samples. Experiments with high dimensional Mountain Car indicate that this behaviour might be due to the fact the state and action space for the basic Gym environments is quite small. Note that we use the same hyperparameters in deterministic and stochastic environments. (Figure 9 (d-i)).

How does the performance of shooting algorithms vary when the size of the action space increases? When the action space increases, a small number of samples cover a very small volume of the possible trajectories. Intuitively, the performance of agents relying on shooting methods will be poor in such scenarios and performance should improve on increasing the number of samples. As shown in Figure 2i, this indeed is the case. We take a closer look at the performance of CEM as the population size increases. For this, we use the modified version of Mountain Car as detailed in Section 4.3 with 14 redundant actions. For this experiment we fix the state and consider action selection in CEM which is done with $K = 10$ optimization steps. We plot the values of 5 action variables (A1 to A5) across 10 optimization steps where A1 influences the dynamics while the rest are redundant action variables. The action value after the 10th optimization step is output by CEM and is executed by the agent in the environment. For attaining high rewards, an agent must learn to keep the values of action variables A2 through A5 as close to 0 as possible, while using A1 to act optimally. The result is shown in Figure 10. We observe that as the population size increases, CEM does a better job at keeping the values of the redundant variables close to 0. Plots shown in the main paper show that this indeed corresponds to improved performance.

Does DiSProD provide a better approximation of the system dynamics than DiSProD-NV? As discussed earlier, DiSProD-NV does not account for any uncertainty in the dynamics and ignores the variance terms in the Taylor's expansion while DiSProD uses a 2nd degree Taylor's expansion to approximate the dynamics. Note that DiSProD-NV does not calculate a distribution but simply propagates the mean. In addition, its calculated mean is potentially less accurate. We explore this behaviour in SimpleEnv. For a fixed sequence of action distributions, we compute the state distribution due to DiSProD and DiSProD-NV and compare that to the empirical state distribution visited by the agent when it samples from the same action distribution. In Figure 11 we compare the empirical state distribution of x against the state distributions as computed by DiSProD and DiSProD-NV. When α is 0, the approximation using DiSProD overlaps with the empirical distribution for some time before diverging slightly. As the stochasticity of the environment increases (indicated by



(a) CCP ($\alpha = 0$) (b) P ($\alpha = 0$) (c) CMC ($\alpha = 0$)

(d) CCP ($\alpha = 0$) (e) P ($\alpha = 0$) (f) CMC ($\alpha = 0$)

(g) CCP ($\alpha = 5$) (h) P ($\alpha = 2$) (i) CMC ($\alpha = 0.002$)

Figure 9: Environments used are Continuous Cartpole (CCP), Pendulum (P) and Continuous Mountain Car (CMC). 9a, 9b, 9c: In deterministic environments ($\alpha = 0$), optimal performance is achieved with a small planning horizon. Increasing the planning horizon further does not impact performance of any of the planners. 9d, 9e, 9f: A similar behaviour is observed when number of samples is varied and $\alpha = 0$. 9g, 9h, 9i: In noisy environments, DiSProD performs better than CEM and MPPI. Interestingly, increasing the number of samples does not improve the performance. Note that the hyperparameters used are same in both deterministic and stochastic environments.

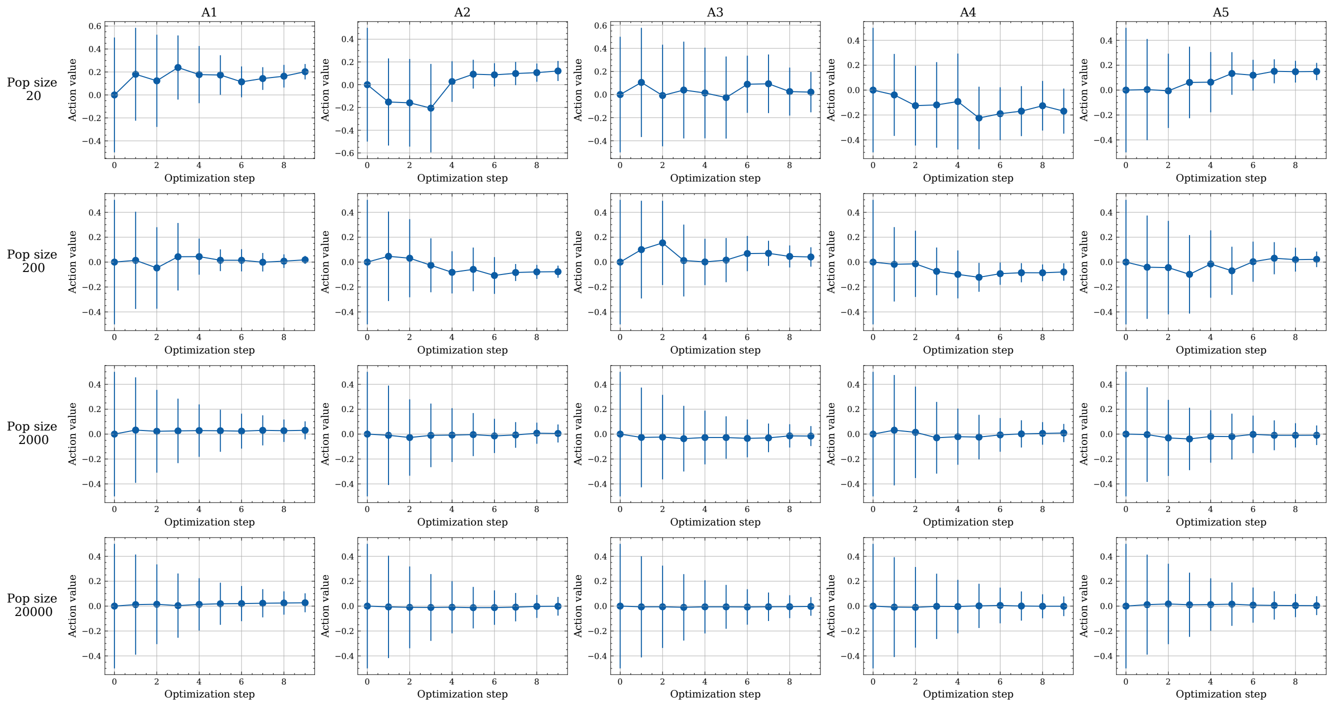


Figure 10: Values of 5 action variables (A1 to A5) across 10 optimization steps as the population size is increased. The action value at the 10th optimization step is executed in the environment. Only A1 influences the dynamics, whereas all of the action variables contribute to action penalty in the reward function. An agent must learn to optimize A1 to achieve the goal, while keeping the values of A2 to A5 close to 0 and achieve a higher reward. When the population size is small, CEM fails to achieve this and obtains a poor score.

increasing α values), the state distributions due to DiSProD and DiSProD-NV diverge from the empirical state distribution. But the distribution due to DiSProD is a better approximation than the distribution due to DiSProD-NV.

Do both state and action variance improve planning? DiSProD relies on state and action variance terms to propagate distributions. Intuitively, state variance captures the impact of noise on the state variables while having action variance enables DiSProD to search over a stochastic policy. We explore the performance of the three variants of DiSProD discussed in Appendix B. Results are shown in Figure 12a, 12b, 12c. We observe that for regions of low stochasticity, all the three modes have similar performance. But with increasing α, zeroing out the state variance harms DiSProD. Further, in all of the basic environments, zeroing out action variance but keeping state variance does not harm performance. However, as shown in Figure 12d, in the Simple Env which has a more complex noise structure (see Appendix C.1) action variance leads to further improvement in performance.

Detailed results for experiments with modified Dubins Car model The detailed results for experiments with the modified Dubins Car model in Gym and TurtleBot are given in Tables 4 and 5. These are aggregated in the main paper in Table 1 by averaging over the 16 maps.

E Runtime

We compare the running time of DiSProD and DiSProD-NV with CEM and MPPI in the basic Gym environments on a system with 3.5GHz i7 CPU and 32GB of memory. We run each planner for 5 episodes and compute the mean and standard deviation of the time taken for each episode. As shown in the table below, the run time of DiSProD-NV is close to both CEM and MPPI, while DiSProD is up to 7 times slower. DiSProD computes the diagonal of the Hessian matrix for propagating distribution which is expensive. The partials also add to the cost of the gradient computation while optimizing actions. Both of these computations can be expensive for high-dimensional state spaces.

Environment	CEM	MPPI	DiSProD-NV	DiSProD
Cart Pole	3.69 ± 0.68	3.70 ± 0.75	4.10 ± 1.60	22.26 ± 4.50
Pendulum	3.68 ± 0.68	3.72 ± 0.78	4.11 ± 1.54	18.16 ± 6.73
Mountain Car	2.04 ± 0.76	2.24 ± 0.84	2.66 ± 1.69	5.70 ± 2.93

Map	CEM		MPPI		DiSProD	
	%Success$\uparrow_0^{100}$	#Steps$\downarrow$	%Success$\uparrow_0^{100}$	#Steps$\downarrow$	%Success$\uparrow_0^{100}$	#Steps$\downarrow$
no-ob-1	100.0 ± 0.0	37.0 ± 0.4	100.0 ± 0.0	38.0 ± 0.0	100.0 ± 0.0	$\mathbf{35.0 \pm 1.6}$
no-ob-2	100.0 ± 0.0	44.0 ± 0.0	100.0 ± 0.0	48.0 ± 0.0	100.0 ± 0.0	$\mathbf{42.0 \pm 1.2}$
no-ob-3	100.0 ± 0.0	$\mathbf{63.0 \pm 0.0}$	100.0 ± 0.0	67.0 ± 0.8	100.0 ± 0.0	$\mathbf{63.0 \pm 0.0}$
no-ob-4	100.0 ± 0.0	$\mathbf{44.0 \pm 0.0}$	100.0 ± 0.0	49.0 ± 0.0	100.0 ± 0.0	$\mathbf{44.0 \pm 0.0}$
no-ob-5	100.0 ± 0.0	39.0 ± 2.0	100.0 ± 0.0	37.0 ± 0.4	100.0 ± 0.0	$\mathbf{30.0 \pm 0.0}$
ob-1	100.0 ± 0.0	$\mathbf{22.0 \pm 0.0}$	100.0 ± 0.0	29.0 ± 0.0	100.0 ± 0.0	$\mathbf{22.0 \pm 0.8}$
ob-2	100.0 ± 0.0	77.0 ± 2.71	100.0 ± 0.0	76.0 ± 2.86	100.0 ± 0.0	$\mathbf{75.0 \pm 2.0}$
ob-3	100.0 ± 0.0	36.0 ± 2.0	100.0 ± 0.0	39.0 ± 0.4	100.0 ± 0.0	$\mathbf{31.0 \pm 1.6}$
ob-4	100.0 ± 0.0	$\mathbf{41.0 \pm 0.0}$	100.0 ± 0.0	46.0 ± 0.0	100.0 ± 0.0	45.0 ± 6.4
ob-5	100.0 ± 0.0	$\mathbf{47.0 \pm 0.0}$	100.0 ± 0.0	$\mathbf{47.0 \pm 0.0}$	100.0 ± 0.0	49.0 ± 0.0
ob-6	$\mathbf{100.0 \pm 0.0}$	$\mathbf{60.0 \pm 0.0}$	80.0 ± 40.0	133.0 ± 133.33	$\mathbf{100.0 \pm 0.0}$	$\mathbf{60.0 \pm 0.0}$
ob-7	100.0 ± 0.0	$\mathbf{32.0 \pm 0.0}$	100.0 ± 0.0	38.0 ± 0.0	100.0 ± 0.0	$\mathbf{31.0 \pm 3.72}$
ob-8	100.0 ± 0.0	44.0 ± 0.0	100.0 ± 0.0	48.0 ± 2.0	100.0 ± 0.0	$\mathbf{39.0 \pm 0.8}$
ob-9	100.0 ± 0.0	$\mathbf{31.0 \pm 0.0}$	100.0 ± 0.0	$\mathbf{32.0 \pm 0.0}$	100.0 ± 0.0	42.0 ± 0.0
ob-10	100.0 ± 0.0	46.0 ± 4.0	100.0 ± 0.0	41.0 ± 2.4	100.0 ± 0.0	$\mathbf{29.0 \pm 0.0}$
ob-11	100.0 ± 0.0	$\mathbf{45.0 \pm 0.0}$	100.0 ± 0.0	48.0 ± 0.4	100.0 ± 0.0	48.0 ± 0.0
cave-mini	100.0 ± 0.0	79.0 ± 1.6	100.0 ± 0.0	85.0 ± 0.0	100.0 ± 0.0	$\mathbf{75.0 \pm 4.8}$

Table 4: Success rate and number of steps taken by the planners for each map when run in the Gym environment. The dynamics model used by the planners is an accurate representation of the true dynamics of the system. In most maps, DiSProD takes the least number of steps to reach the goal.

Maps	CEM		MPPI		DiSProD	
	%Success$\uparrow_0^{100}$	#Steps$\downarrow$	%Success$\uparrow_0^{100}$	#Steps$\downarrow$	%Success$\uparrow_0^{100}$	#Steps$\downarrow$
no-ob-1	100.0 ± 0.0	83.0 ± 12.2	100.0 ± 0.0	134.0 ± 90.04	100.0 ± 0.0	$\mathbf{59.0 \pm 9.56}$
no-ob-2	100.0 ± 0.0	$\mathbf{169.0 \pm 6.95}$	100.0 ± 0.0	180.0 ± 3.88	100.0 ± 0.0	174.0 ± 14.05
no-ob-3	100.0 ± 0.0	163.0 ± 7.84	100.0 ± 0.0	195.0 ± 7.22	100.0 ± 0.0	$\mathbf{143.0 \pm 16.77}$
no-ob-4	100.0 ± 0.0	116.0 ± 3.88	100.0 ± 0.0	121.0 ± 2.87	100.0 ± 0.0	$\mathbf{101.0 \pm 12.67}$
no-ob-5	100.0 ± 0.0	$\mathbf{197.0 \pm 6.08}$	100.0 ± 0.0	203.0 ± 5.12	100.0 ± 0.0	211.0 ± 14.18
ob-1	40.0 ± 49.0	289.0 ± 136.44	$\mathbf{100.0 \pm 0.0}$	$\mathbf{149.0 \pm 2.1}$	$\mathbf{100.0 \pm 0.0}$	155.0 ± 51.79
ob-2	100.0 ± 0.0	89.0 ± 2.86	100.0 ± 0.0	120.0 ± 2.42	100.0 ± 0.0	$\mathbf{88.0 \pm 11.62}$
ob-3	100.0 ± 0.0	$\mathbf{96.0 \pm 4.35}$	100.0 ± 0.0	136.0 ± 4.8	100.0 ± 0.0	97.0 ± 5.68
ob-4	100.0 ± 0.0	$\mathbf{132.0 \pm 10.13}$	100.0 ± 0.0	177.0 ± 8.52	100.0 ± 0.0	144.0 ± 16.81
ob-5	100.0 ± 0.0	$\mathbf{93.0 \pm 7.91}$	100.0 ± 0.0	149.0 ± 3.61	100.0 ± 0.0	103.0 ± 6.77
ob-6	0.0 ± 0.0	400.0 ± 0.0	0.0 ± 0.0	400.0 ± 0.0	$\mathbf{80.0 \pm 40.0}$	$\mathbf{243.0 \pm 115.9}$
ob-7	100.0 ± 0.0	$\mathbf{87.0 \pm 5.23}$	100.0 ± 0.0	127.0 ± 4.59	100.0 ± 0.0	94.0 ± 4.83
ob-8	100.0 ± 0.0	$\mathbf{133.0 \pm 1.36}$	100.0 ± 0.0	175.0 ± 14.43	100.0 ± 0.0	155.0 ± 9.4
ob-9	100.0 ± 0.0	112.0 ± 7.24	100.0 ± 0.0	$\mathbf{110.0 \pm 2.68}$	100.0 ± 0.0	126.0 ± 33.85
ob-10	40.0 ± 49.0	298.0 ± 125.42	$\mathbf{100.0 \pm 0.0}$	$\mathbf{150.0 \pm 6.09}$	60.0 ± 49.0	233.0 ± 137.32
ob-11	$\mathbf{80.0 \pm 40.0}$	239.0 ± 80.75	60.0 ± 49.0	275.0 ± 102.4	$\mathbf{80.0 \pm 40.0}$	$\mathbf{230.0 \pm 88.32}$
cave-mini	100.0 ± 0.0	$\mathbf{290.0 \pm 15.97}$	0.0 ± 0.0	400.0 ± 0.0	100.0 ± 0.0	332.0 ± 20.43

Table 5: Success rate and number of steps taken by the planners for each map while controlling Turtlebot. The dynamics model used by the planners in an approximation of the true dynamics of the system. Although in some cases DiSProD takes longer to complete the goal in maps with obstacles, it has a better overall SR as compared to CEM/MPPI.

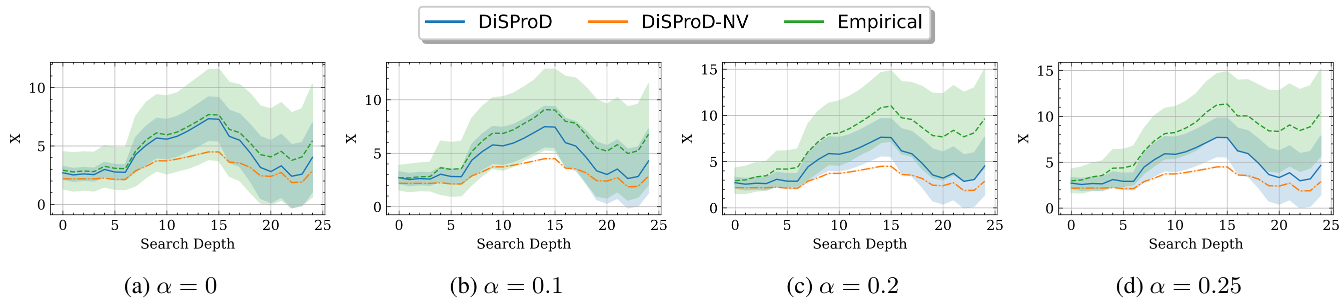


Figure 11: For a fixed action distribution, the state distribution simulated by DSSPD, while planning in Simple Env, is a better approximation of the empirical state distribution than the one due to DSSPD-NV.

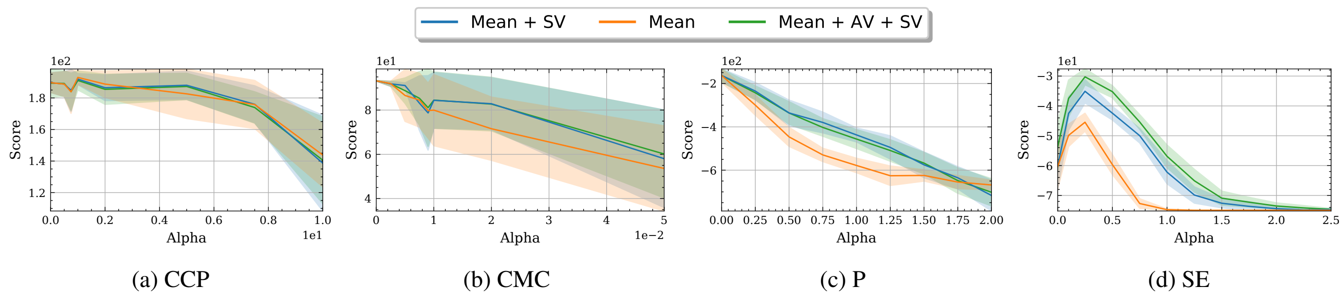


Figure 12: Environments used are Continuous Cartpole (CCP), Pendulum (P), Continuous Mountain Car (CMC) and Simple Env (SE). Ignoring the variance terms generally hurts DiSProD, especially with increasing α. While action variance does not contribute a lot to the planner's performance in some environments (a, b, c), but in others (d) searching over a stochastic policy yields further improvement.