INFORMATION THEORETIC MODEL PREDICTIVE Q-LEARNING

Mohak Bhardwaj[1,2] **Ankur Handa**[2] **Dieter Fox**[1,2] **Byron Boots**[1,2]

[1] **University of Washington** [2] **NVIDIA**

ABSTRACT

Model-free Reinforcement Learning (RL) algorithms work well in sequential decision-making problems when experience can be collected cheaply and model-based RL is effective when system dynamics can be modeled accurately. However, both of these assumptions can be violated in real world problems such as robotics, where querying the system can be prohibitively expensive and real-world dynamics can be difficult to model accurately. Although sim-to-real approaches such as domain randomization attempt to mitigate the effects of biased simulation, they can still suffer from optimization challenges such as local minima and hand-designed distributions for randomization, making it difficult to learn an accurate global value function or policy that directly transfers to the real world. In contrast to RL, Model Predictive Control (MPC) algorithms use a simulator to optimize a simple policy class online, constructing a closed-loop controller that can effectively contend with real-world dynamics. MPC performance is usually limited by factors such as model bias and the limited horizon of optimization. In this work, we present a novel theoretical connection between information theoretic MPC and entropy regularized RL and develop a Q-learning algorithm that can leverage biased models. We validate the proposed algorithm on sim-to-sim control tasks to demonstrate the improvements over optimal control and reinforcement learning from scratch. Our approach paves the way for deploying reinforcement learning algorithms on real-robots in a systematic manner.

1 INTRODUCTION

Deep reinforcement learning algorithms have recently generated great interest due to their successful application to a range of difficult problems including Computer Go [26] and high-dimensional control tasks such as humanoid locomotion [14, 22]. While these methods are extremely general and can learn policies and value functions for complex tasks directly from raw data, they can also be sample inefficient, and partially-optimized solutions can be arbitrarily poor. These challenges severely restrict RL's applicability to real systems such as robots due to data collection challenges and safety concerns.

One straightforward way to mitigate these issues is to learn a policy or value function entirely in a high-fidelity simulator [24, 34] and then deploy the optimized policy on the real system. However, this approach can fail due to model bias, external disturbances, the subtle differences between the real robot's hardware and poorly modeled phenomena such as friction and contact dynamics. Sim-to-real transfer approaches based on domain randomization [21, 32] and model ensembles [13, 25] aim to make the policy robust by training it to be invariant to varying dynamics. However, learning a globally consistent value function or policy is hard due to optimization issues such as local optima and covariate shift between the exploration policy used for learning the model and the actual control policy executed on the task [20].

Model predictive control (MPC) is a widely used method for generating feedback controllers that repeatedly re-optimizes a finite horizon sequence of controls using an approximate dynamics model that predicts the effect of these controls on the system. The first control in the optimized sequence is executed on the real system and the optimization is performed again from the resulting next state. However, the performance of MPC can suffer due to approximate or simplified models and a limited lookahead. Therefore the parameters of MPC, including the model and horizon H need to be carefully tuned to obtain good performance. While using a longer horizon is generally preferred, real-time requirements may limit the amount of lookahead and a biased model can result in compounding model errors.

In this work, we present an approach to RL that leverages the complementary properties of model-free reinforcement learning and model-based optimal control. Our proposed method views MPC as a way to simultaneously approximate and optimize a local Q function via simulation, and Q learning as a way to improve MPC using real-world data. We focus on the paradigm of entropy regularized reinforcement learning where the aim is to learn a stochastic policy that minimizes the cost-to-go as well as KL divergence with respect to a prior policy. This approach enables faster convergence by mitigating the over-commitment issue in the early stages of Q-learning and better exploration [7]. We discuss how this formulation of reinforcement learning has deep connections to information theoretic stochastic optimal control where the objective is to find control inputs that minimize the cost while staying close to the passive dynamics of the system [31]. This helps in both injecting domain knowledge into the controller as well as mitigating issues caused by over optimizing the biased estimate of the current cost due to model error and the limited horizon of optimization. We explore this connection in depth and derive an infinite horizon information theoretic model predictive control algorithm based on Williams et al. [38]. We test our approach called Model Predictive Q Learning (MPQ) on simulated continuous control tasks and compare it against information theoretic MPC and soft Q-Learning [9], where we demonstrate faster learning with fewer system interactions and better performance as compared to MPC and soft Q-Learning even in the presence of sparse rewards. The learned Q function allows us to truncate the MPC planning horizon which provides additional computational benefits. Finally, we also compare MPQ versus domain randomization(DR) on sim-to-sim tasks. We conclude that DR approaches can be sensitive to the hand-designed distributions for randomizing parameters which causes the learned Q function to be biased and suboptimal on the true system's parameters, whereas learning from data generated on true system is able to overcome biases and adapt to the real dynamics.

2 Related Work

Model predictive control has a rich history in robotics, ranging from control of mobile robots such as quadrotors [5] and aggressive autonomous vehicles [37, 38] to generating complex behaviors for high-dimensional systems such as contact-rich manipulation [8, 12] and humanoid locomotion [6]. The success of MPC can largely be attributed to online policy optimization which helps mitigate model bias. The information theoretic view of MPC aims to find a policy at every timestep that minimizes the cost over a finite horizon as well as the KL-divergence with respect to a prior policy usually specified by the system's passive dynamics [31, 38]. This helps maintain exploratory behavior and avoid over-commitment to the current estimate of the cost function, which is biased due to modeling errors and a finite horizon. Sampling-based MPC algorithms [37, 38] are also highly parallelizable enabling GPU implementations that aid with real-time control. However, efficient MPC implementations still require careful system identification and extensive amounts of manual tuning.

Deep RL methods are extremely general and can optimize neural network policies from raw sensory inputs with little knowledge of the system dynamics. Both value-based and policy-based approaches [22] have demonstrated excellent performance on complex control problems. These approaches, however, fall short on several accounts when applying them to a real robotic system. First, they have high sample complexity, potentially requiring millions of interactions with the environment. This can be very expensive on a real robot, not least because the initial performance of the policy can be arbitrarily bad. Using random exploration methods such a ϵ-greedy can further aggravate this problem. Second, a value function or policy learned entirely in simulation inherits the biases of the simulator. Even if a perfect simulation is available, learning a globally consistent value function or policy is an extremely hard task as noted in [26, 39]. This can be attributed to local optima when using neural network representations or the inherent biases in the Q learning update rules [7, 35]. In fact, it can be difficult to explain why Q-learning algorithms work or fail [23].

Domain randomization aims to make policies learned in simulation more robust by randomizing simulation parameters during training with the aim of making the policies invariant to potential parameter error [17, 21, 32]. However, these policies are not *adaptive* to unmodelled effects, i.e they take into account only aleoteric and not epistemic uncertainty. Also, such approaches are highly sensitive to hand-designed distributions used for randomizing simulation parameters and can be highly suboptimal on the real-systems parameters, for example, if a very large range of simulation parameters is used. Model-based approaches aim to use real data to improve the model of the system and then perform reinforcement learning or optimal control using the new model or ensemble of models [13, 20, 25]. Although learning accurate models is a promising avenue, we argue that learning a globally consistent model is an extremely hard problem and instead we should learn a policy that can rapidly adapt to experienced real-world dynamics.

The use of entropy regularization has been explored in RL and Inverse RL for its better sample efficiency and exploration properties [7, 9, 10, 23, 40]. This framework allows incorporating prior knowledge into the problem and learning multi-modal policies that can generalize across different tasks. Fox et al. [7] analyze the theoretical properties of the update rule derived using mutual information minimization and show that this framework can overcome the over-estimation issue inherent in the vanilla Q-learning update. In the past, Todorov [33] have shown that using KL-divergence can convert the optimal control problem into one that is linearly solvable.

Infinite horizon MPC aims to learn a terminal cost function that can add global information to the finite horizon optimization. [19] learn a terminal cost as a control Lyapunov function and a safety set for the terminal state. These quantites are calculated using all previously visited states and they assume the presence of a controller that can deterministically drive the any state to the goal. [29] learns a cost shaping to make a short horizon MPC mimic the actions produced by long horizon MPC offline. However, since their approach is to mimic a longer horizon MPC, the performance of the learner is fundamentally limited by the the performance of the longer horizon MPC. On the contrary, learning an optimal value function as the terminal cost can potentially lead to close to optimal performance.

Using local optimization is an effective way of improving an imperfect value function as noted in RL literature by [1, 15, 26–28]. However, these approaches assume that a perfect model of the system is available. In order to make the policy work on the real system, we argue that it is essential to learn a value function from real data and utilize local optimization to stabilize learning.

3 Preliminaries

We first develop relevant notation and introduce the entropy-regularized RL and information theoretic MPC frameworks. We show that they are complimentary approaches to solve a similar problem.

3.1 Reinforcement Learning with Entropy Regularization

A Markov Decision Process (MDP) is defined by tuple $(\mathcal{S}, \mathcal{A}, c, \mathcal{P}, \gamma)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, c is a one step cost function, P is the transition function and γ is a discount factor. A closed-loop policy $\pi(a|s)$ is a distribution over actions given state. Given a policy π and a *prior* policy $\overline{\pi}$, the KL divergence between them at a state is given by $KL\left(\pi(a|s)||\overline{\pi}(a|s)\right) = \mathbb{E}_{\pi}\left[\log\left(\pi(a|s)/\overline{\pi}(a|s)\right)\right]$. Entropy-regularized RL [7] aims to optimize the objective

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{\pi,P}\left[\sum_{t=1}^{\infty} \gamma^{t-1}\left(c(s_t, a_t) + \lambda KL\left(\pi_t||\overline{\pi}_t\right)\right)\right] \; \forall \; s_0 \; \in \; \mathcal{S} \tag{1}$$

where π_t and $\overline{\pi}_t$ are shorthand for $\pi(a_t|s_t)$ and $\overline{\pi}(a_t|s_t)$ respectively, λ is a temperature parameter that penalizes deviation of π from $\overline{\pi}$. Given π, we can define the *soft* value functions as[1]

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi,P}\left[\sum_{t=1}^{\infty} \gamma^{t-1}\left(c(s_t, a_t) + \lambda KL\left(\pi_t||\overline{\pi}_t\right)\right)|s_0 = s\right] \\ Q^{\pi}(s,a) &= c(s,a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)}\left[V^{\pi}(s')\right] \end{aligned} \tag{2}$$

[1] In this work we consider costs instead of rewards and hence aim to find policies that minimize cumulative cost-to-go.

Given a horizon of H timesteps, we can use above definitions to write the value functions as

$$V^{\pi}(s) = \mathbb{E}_{\pi,P}\left[\sum_{t=1}^{H-1} \gamma^{t-1}(c(s_t,a_t) + \lambda KL\left(\pi_t||\overline{\pi}_t\right)) + \gamma^{H-1}V^{\pi}(s_H)|s_1 = s\right]$$

$$\begin{aligned} Q^{\pi}(s,a) = c(s,a) + \mathbb{E}_{\pi,P}\Bigg[&\sum_{t=2}^{H-1} \gamma^{t-1}(c(s_t,a_t) + \lambda KL\left(\pi_t||\overline{\pi}_t\right)) \\ &+ \gamma^{H-1}(\lambda KL\left(\pi_H||\overline{\pi}_H\right) + Q(s_H,a_H))|s_1 = s, a_1 = a\Bigg] \end{aligned} \tag{3}$$

It is straightforward to verify that $V^{\pi}(s) = \mathbb{E}_{a\sim\pi}\left[\log(\pi(a|s)/\overline{\pi}(a|s)) + Q(s,a)\right]$. The objective in Eq. (1) can equivalently be written as

$$\pi^* = \arg\min_{\pi} V^{\pi}(s)\ \forall\ s\ \in\ \mathcal{S} \tag{4}$$

The above optimization can be performed either by policy gradient methods that aim to find the optimal policy $\pi \in \Pi$ via stochastic gradient descent [23] or value based methods that try to iteratively approximate the value function of the optimal policy. In either case, the output of solving the above optimization is a global closed-loop control policy $\pi^*(a|s)$.

3.2 Information Theoretic MPC

Solving the above optimization can be prohibitively expensive and hard to accomplish *online*, i.e. at every time step as the system executes, especially when using complex policy classes like deep neural networks. In contrast to this approach, MPC performs online optimization of a simple policy class with a truncated horizon. To achieve this, MPC algorithms such as Model Predictive Path Integral Control (MPPI) [38] use an approximate dynamics model $\hat{P}$, which can be a deterministic simulator such as MuJoCo [34]. At timestep t, starting from the current state s_t, an open loop sequence of actions $A = (a_t, a_{t+1}, \ldots a_{t+H})$ is sampled from the control distribution denoted by $\pi(A)$. The objective is to find an optimal sequence of actions to solve

$$\begin{aligned} A^* = \arg\min_{A} \mathbb{E}_{\pi(A)}\Bigg[&\sum_{l=t}^{t+H-1} \gamma^{l-t}(c(s_l,a_l) + \lambda KL\left(\pi_l||\overline{\pi}_l\right)) \\ &+ \gamma^{H-1}(c_f(s_{t+H},a_{t+H}) + \lambda KL\left(\pi_{t+H}||\overline{\pi}_{t+H}\right))\Bigg] \end{aligned} \tag{5}$$

where $c_f(s_{t+H}, a_{t+H})$ is a terminal cost function and $\overline{\pi}(A)$ is the passive dynamics of the system, i.e the distribution over actions produced when the control input is zero. The first action in the sequence is then executed on the system and the optimization is performed again from the resulting next state MB: effectively resulting in a closed-loop controller. The re-optimization and entropy regularization helps in mitigating MB: effects of model-bias and inaccuracies with optimization by avoiding overcommitment to the current estimate of the cost. A shortcoming of the MPC procedure is the finite horizon. This is especially pronounced in tasks with sparse rewards where a short horizon can make the agent myopic to future rewards. In order to mitigate this, an approach known as infinite horizon MPC sets the terminal cost c_f as a value function that adds global information to the problem.

Having introduced the fundamental concepts, in the next section we develop our approach to combine entropy regularized RL with information theoretic MPC and derive the MPPI update rule from Williams et al. [38] for the infinite horizon case.

4 Approach

Infinite-horizon MPC [39] replaces the terminal cost by a value function to add global information to the finite-horizon optimization. We focus on MPPI [38], and show that it implicitly optimizes an upper-bound on the entropy-regularized objective and derive the infinite horizon update rule. We start by deriving the expression for the optimal policy, which is intractable to sample and then a scheme to iteratively approximate it with a simple policy class similar to Williams et al. [38]. Unlike previous approaches, we argue that learning a value function from real system parameters is necessary to mitigate effects of model error.

4.1 Optimal H-step Boltzmann Distribution

Let $\pi(A)$ and $\overline{\pi}(A)$ be the joint open-loop control distribution and prior over H-horizon open-loop actions respectively and π_t is shorthand for $\pi(a_t)$. Since $\hat{P}$ is deterministic, the following holds

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\lambda \log(\pi/\overline{\pi}) + Q^{\pi}(s,a)\right] \qquad Q^{\pi}(s,a) = c(s,a) + \gamma V^{\pi}(s') \tag{6}$$

For clarity, we omit γ. Substituting from Eq. (3) for $Q^{\pi}(s,a)$

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi_1 \dots \pi_H}\left[\sum_{t=1}^{H-1} c(s_t,a_t) + \lambda \sum_{t=1}^{H} \log(\pi_t/\overline{\pi}_t) + Q^{\pi}(s_H,a_H)\right] \\ &= \mathbb{E}_{\pi_1 \dots \pi_H}\left[\sum_{t=1}^{H-1} c(s_t,a_t) + \lambda \log \prod_{t=1}^{H} (\pi_t/\overline{\pi}_t) + Q^{\pi}(s_H,a_H)\right] \\ &\leq \mathbb{E}_{\pi}\left[\sum_{t=1}^{H-1} c(s_t,a_t) + \lambda \log(\pi/\overline{\pi}) + Q^{\pi}(s_H,a_H)\right] \end{aligned} \tag{7}$$

where the final inequality results from replacing product of marginals by the joint distributions. Now, consider the following distribution over H-horizon

$$\pi = \frac{1}{\eta} \exp\left(\frac{-1}{\lambda}\left(\sum_{t=1}^{H-1} c(s_t,a_t) + Q^{\pi}(s_H,a_H)\right)\right)\overline{\pi}(a_1 \dots a_H) \tag{8}$$

where η is a normalizing constant given by

$$\eta = \mathbb{E}_{\overline{\pi}(a_1 \dots a_H)}\left[\exp\left(\frac{-1}{\lambda}\left(\sum_{t=1}^{H-1} c(s_t,a_t) + Q^{\pi}(s_H,a_H)\right)\right)\right] \tag{9}$$

We show that this is the optimal control distribution as $\nabla V^{\pi}(s) = 0$. Substituting Eq. (8) in (7)

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left[\sum_{t=1}^{H-1} c(s_t,a_t) - \lambda \log \eta - \sum_{t=1}^{H-1} c(s_t,a_t) - Q^{\pi}(s_H,a_H) + Q^{\pi}(s_H,a_H)\right] \\ V^{\pi}(s) &= \mathbb{E}_{\pi}\left[-\lambda \log \eta\right] \end{aligned}$$

Since η is a constant, we have $V^{\pi}(s) = -\lambda \log \eta$. Hence for π in Eq. (8), the soft value function is a constant with gradient zero and is given by

$$V^{\pi^*}(s) = -\lambda \log \mathbb{E}_{\overline{\pi}(a_1 \dots a_H)}\left[\exp\left(\frac{-1}{\lambda}\left(\sum_{t=1}^{H-1} c(s_t,a_t) + Q^{\pi}(s_H,a_H)\right)\right)\right] \tag{10}$$

which is often referred to in optimal control literature as the “free energy” of the system [31, 38]. For H=1, Eq. (10) takes the form of the soft value function from [7, 10]. We note that the inequality in Eq. (7) implies that the optimal distribution only optimizes an upper bound to the entropy-regularized objective. This provides the insight that optimal control algorithms such as MPPI that use this distribution have a fundamental performance limit. We wish to further investigate this in future work.

4.2 Infinite Horizon MPPI Update Rule

Similar to [38], we derive the MPPI update rule which is used for online policy optimization. Since sampling actions from the optimal control distribution in Eq. (8) is intractable, we consider control policies $\pi(A) \in \Pi$ which are easy to sample from. We then optimize for a vector of H control inputs U, such that the resulting action distribution minimizes the KL divergence with the optimal policy

$$U^* = \underset{\pi(A)}{\arg\min}\; KL\left(\pi^*(A)||\pi(A)\right) \tag{11}$$

The objective can be expanded out as

$$\begin{aligned} KL\left(\pi^*(A)||\pi(A)\right) &= \int\limits_A \pi^*(A)\log\frac{\pi^*(A)}{\pi(A)}\mathrm{d}A = \int\limits_A \pi^*(A)\log\frac{\pi^*(A)}{\bar{\pi}(A)}\frac{\bar{\pi}(A)}{\pi(A)}\mathrm{d}A \\ &= \int\limits_A \pi^*(A)\log\frac{\pi^*(A)}{\bar{\pi}(A)}\mathrm{d}A - \int\limits_A \pi^*(A)\log\frac{\bar{\pi}(A)}{\pi(A)}\;\mathrm{d}A \end{aligned} \tag{12}$$

Since the first term does not depend on the control input, we can remove it from the optimization

$$U^* = \underset{\pi(A)}{\arg\max} \int\limits_A \pi^*(A)\log\frac{\bar{\pi}(A)}{\pi(A)}\;\mathrm{d}A \tag{13}$$

Consider Π to be independent multivariate Gaussians over sequence of the H controls with constant covariance Σ at each timestep. We can write the control distribution and prior as follows

$$\pi(A) = \frac{1}{Z}\prod_{t=1}^{H}\exp\left(-\frac{1}{2}\left(u_t - a_t\right)^T\Sigma^{-1}\left(u_t - a_t\right)\right) \quad \bar{\pi}(A) = \frac{1}{Z}\prod_{t=1}^{H}\exp\left(-\frac{1}{2}a_t^T\Sigma^{-1}a_t\right) \tag{14}$$

where u_t and a_t are the control inputs and actions respectively at timestep t and Z is the normalizing constant. Here the prior corresponds to the passive dynamics of the system [31, 38], although other choices of prior are possible. Substituting in Eq. (13) we get

$$U^* = \underset{\pi(A)}{\arg\max} \int \;\pi^*(A)\left(\sum_{t=1}^{H} -\frac{1}{2}u_t^T\Sigma^{-1}u_t + u_t^T\Sigma^{-1}a_t\right)\;\mathrm{d}A \tag{15}$$

The objective can be simplified to the following by integrating out the probability in the first term

$$\sum_{t=1}^{H} -\frac{1}{2}u_t^T\Sigma^{-1}u_t + u_t^T\int \pi^*(A)\Sigma^{-1}a_t\;\mathrm{d}A \tag{16}$$

Since this is a concave function with respect to every u_t, we can find the maximum by setting its gradient with respect to u_t to zero to solve for optimal u_t^*

$$u_t^* = \int \pi^*(A)a_t\mathrm{d}A = \int \pi(A)\frac{\pi^*(A)}{\bar{\pi}(A)}\frac{\bar{\pi}(A)}{\pi(A)}a_t\mathrm{d}A = \mathbb{E}_{\pi(A)}\left[\frac{\pi^*(A)}{\bar{\pi}(A)}\frac{\bar{\pi}(A)}{\pi(A)}a_t\right] = \mathbb{E}_{\pi(A)}\left[w(A)a_t\right] \tag{17}$$

where the second equality comes from importance sampling to convert the optimal controls into an expectation over the control distribution instead of the optimal distribution which is impossible to sample from. The importance weight $w(A)$ can be written as follows (substituting π^* from Eq. (8))

$$w(A) = \frac{1}{\eta}\;\mathbb{E}_{\pi(A)}\left[\exp\left(\frac{-1}{\lambda}\left(\sum_{t=1}^{H-1} c(s_t, a_t) + Q^{\pi^*}(s_H, a_H)\right)\right)\frac{\bar{\pi}(A)}{\pi(A)}\right] \tag{18}$$

Making change of variables $u_t + \epsilon_t = a_t$ for noise sequence $\mathcal{E} = (\epsilon_1 \ldots \epsilon_H)$ sampled from independant Gaussians with zero mean and covariance Σ we get

$$\begin{aligned} w(\mathcal{E}) &= \frac{1}{\eta}\;\mathbb{E}_{\pi(A)}\left[\exp\left(\frac{-1}{\lambda}\left(\sum_{t=1}^{H-1} c(s_t, u_t + \epsilon_t) + \lambda\frac{\pi(U + \mathcal{E})}{\bar{\pi}(U + \mathcal{E})} + Q^{\pi^*}(s_H, u_H + \epsilon_H)\right)\right)\right] \\ &= \frac{1}{\eta}\;\mathbb{E}_{\pi(A)}\left[\exp\left(\frac{-1}{\lambda}\left(\sum_{t=1}^{H-1} c(s_t, u_t + \epsilon_t) + \lambda\sum_{t=1}^{H}\frac{1}{2}u_t^T\Sigma^{-1}(u_t + 2\epsilon_t) + Q^{\pi^*}(s_H, u_H + \epsilon_H)\right)\right)\right] \end{aligned} \tag{19}$$

Note that η is the optimal H-step free energy derived in Eq. (10) and can be estimated from N Monte-Carlo samples as

$$\eta = \sum_{n=1}^{N} \exp\left(\frac{-1}{\lambda}\left(\sum_{t=1}^{H-1} c(s_t, u_t + \epsilon_t^n) + \lambda \sum_{t=1}^{H} \frac{1}{2} u_t^T \Sigma^{-1}(u_t + 2\epsilon_t^n) + Q^{\pi^*}(s_H, u_H + \epsilon_H^n)\right)\right) \tag{20}$$

We can form the following iterative update rule where at every iteration i the sampled control sequence is updated according to

$$u_t^{i+1} = u_t^i + \alpha \sum_{n=1}^{N} w(\mathcal{E}_n)\epsilon_n \tag{21}$$

where α is a step-size parameter as proposed by [37]. This gives us the infinite horizon MPPI update rule. For $H = 1$, this corresponds soft Q-learning where stochastic optimization is performved to solve for the optimal action online. Now we develop soft Q-learning algorithm that utilizes infinite horizon MPPI to generate actions as well as Q-targets.

4.3 Information Theoretic Model Predictive Q-Learning Algorithm

We consider Q functions parameterized by θ denoted by $Q_\theta(s, a)$ and update parameters by stochastic gradient descent on the loss $L = \frac{1}{K}\sum_{i=1}^{K}(y_i - Q_\theta(s_i, a_i))^2$ for a batch of K experience tuples (s, a, c, s') sampled from a replay buffer [16] where targets y_i are given by

$$y = c(s, a) - \gamma\lambda \log \mathbb{E}_{\pi^*(a_1 \ldots a_H)}\left[\exp\left(\frac{-1}{\lambda}\left(\sum_{t=1}^{H-1} c(s_t, a_t) + \sum_{t=1}^{H} \lambda \log \frac{\pi_t^*}{\overline{\pi}_t} + Q_\theta(s_H, a_H)\right)\right)\middle| s_1 = s'\right] \tag{22}$$

Since the value function updates are performed offline, we can utilize large amounts of computation [29] to calculate $\pi^*(a_1 \ldots a_H)$. We do so by performing multiple iterations of the infinite horizon MPPI update in Eq. (21) from s', which allows for directed exploration and better approximation of the free energy (akin to approaches such as Covariance Matrix Adaption, although MPPI does not adapt the covariance). This helps in early stages of learning by providing better quality targets than a random Q function. Intuitively, this update rule leverages the biased dynamics model $\hat{P}$ for H steps and a soft Q function at the end learned from interactions with the real system.

At every timestep t during online rollouts, an H-horizon sequence of actions is optimized using infinite horizon MPPI and the first action is executed on the system. Online optimization with predictive models can look ahead to produce better actions making ad-hoc exploration strategies such as ϵ-greedy unnecessary. Using predictive models for generating value targets and online policy optimization helps accelerate convergence as we demonstrate in our experiments in the next section. Algorithm 1 shows the complete MPQ algorithm. A closely related approach in literature is POLO Lowrey et al. [15], which also uses MPPI and offline value function learning, however POLO assumes access to the true dynamics and does not explore the connection between MPPI and entropy regularized RL, and thus does not use free energy targets.

5 Experiments

We evaluate the efficacy of MPQ on two fronts: (a) overcoming the shortcomings of both stochastic optimal control and model free RL in terms of computational requirements, model bias, and sample efficiency; and (b) learning effective policies on systems for which accurate models are not known.

5.1 Experimental Setup

We focus on sim-to-sim continuous control tasks using the Mujoco simulator [34] (except PendulumSwingup that uses dynamics equations) to study the properties of our algorithm in a controlled manner. We consider robotics-inspired tasks with either sparse rewards or requiring long-horizon planning. The complexity is further aggravated as the agent is not provided with the true dynamics parameters, but rather a uniform distribution over them with a biased mean and added noise. Details of the tasks considered are as follows

Algorithm 1: MPQ

```
Input    : Approximate model $\hat{P}$, initial Q function parameters $\theta_1$, experience buffer $\mathcal{D}$
Parameter: Number of episodes $N$, length of episode $T$, planning horizon $H$, number of
           update episodes $N_{update}$, minibatch-size $K$, number of minibatches $M$
for $i = 1 \ldots N$ do
    for $t = 1 \ldots T$ do
        $(a_t, \ldots, a_{t+H}) \leftarrow$ Infinite horizon MPPI (Eq. (21))
        Execute $a_t$ on the real system to obtain $c(s_t, a_t)$ and next state $s_{t+1}$
        $\mathcal{D} \leftarrow (s_t, a_t, c, s_{t+1})$
    if $i \% N_{update} == 0$ then
        Sample $M$ minibatches of size $K$ from $\mathcal{D}$
        Generate targets using Eq. (22) and update parameters to $\theta_{i+1}$
    return $\theta_N$ or best $\theta$ on validation.
```

1. PendulumSwingup: the agent tries to swingup and stabilize a pendulum by applying torque on the hinge given a biased distribution over its mass and length. The cost penalizes the deviation from the upright position and angular velocity. Initial state is randomized after every episode of 10s.

2. BallInCupSparse: a sparse version of the task from the Deepmind Control Suite Tassa et al. [30]. Given a cup and ball attached by a tendon, the goal is to swing and catch the ball. The agent controls motors on the two slide joints on the cup and is provided with a biased distribution over the ball's mass, moment of inertia and tendon stiffness. A cost of 1 is incurred at every timestep and 0 if the ball is in the cup which corresponds to success. The position of the ball is randomized after every episode, which is 4 seconds long.
3. FetchPushBlock: proposed by Plappert et al. [18], the agent controls the cartesian position and opening a Fetch robot gripper to push a block to a goal location. The cost is the distance between the center of mass of the block and the goal. We provide the agent a biased distribution over the mass, moment of inertia, friction coefficients and size of the object. An episode is successful if the agent gets the block within 5cm of the goal in 4 seconds. The positions of both block and goal are randomized after every episode.
4. FrankaDrawerOpen: based on a real-world manipulation problem from [3] where the agent velocity controls a 7DOF Franka Panda arm to open a cabinet drawer. A simple cost function based on Euclidean distance and relative orientation of the end effector with respect to the handle and the displacement of the slide joint on the drawer is used. A biased distribution over damping and frictionloss of robot and drawer joints is provided. Every episode is 4 seconds long after which the arm's start configuration is randomized. Success corresponds to opening the drawer within 1cm of a target displacement.

We used BallInCupSparse, FetchPushBlock and FrankaDrawerOpen because they are more realistic proxies for real-world robotics tasks as compared to standard OpenAI Gym [2] baselines such as Ant and HalfCheetah. The parameters we randomize are reasonable in real world scenarios as estimating moment inertia and friction coefficients is especially error prone. Details of default parameters and randomization distributions are in Table 1. All experiments were performed on a desktop with 12 Intel Core i7-3930K @ 3.20GHz CPUs and 32 GB RAM with only few hours of CPU training. Q-functions are parameterized with feed-forward neural networks that take as input an observation vector and action. Refer to A.1 for detailed explanation of tasks.

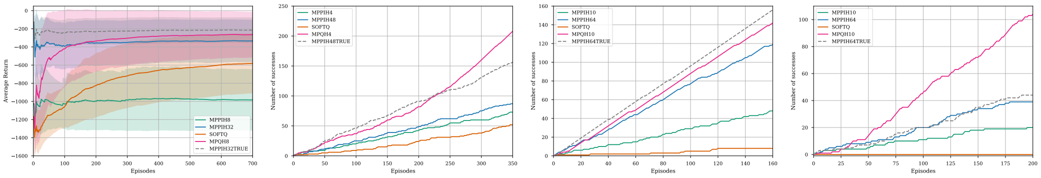
5.2 Analysis of Overall Performance

By learning a terminal value function from real data we posit that MPQ will adapt to true system dynamics and truncate the horizon of MPC by adding global information. Using MPC for Q targets, we also expect to be able to learn with significantly less data as compared to model-free soft Q-learning. Hence, we compare MPQ with the following natural baselines: MPPI using same horizon as MPQ and no terminal value function, MPPI using a longer horizon, SoftQLearning with target networks. Note that MPQ does not use a target network. We do not compare against model-based RL methods [4, 13] as learning globally consistent neural network models adds an additional

Table 1: Environment parameters and dynamics randomization. The last column denotes the range for the uniform distribution. I_{xyz} implies that moment of inertia is the same along all three axes. T is the tendon stiffness. For FetchPushBlock, the block is a cube with sides of length l. FetchPushBlock and FrankaDrawerOpen use uniform distribution for every parameter defined by: mean = bias × true value and range = $[-\sigma \times$ true value, $\sigma \times$ true value$]$

Environment	Cost Function	True Parameters	Biased Distribution
PendulumSwingup	$\Theta^2 + 0.1\dot{\Theta}^2$	$m = 1$kg $l = 1$m	$m = [0.9, 1.5]$ $l = [0.9, 1.5]$
BallInCupSparse	0 if ball in cup 1 else	$m = 0.058$kg $I_{xyz} = 1.47 \times 10^{-5}$ $T = 0.05$	$m = [0.0087, 0.87]$ $I_{xyz} = [0.22, 22] \times 10^{-5}$ $T = [0.00375, 1.5]$
FetchPushBlock	$d_{block,goal}$	$m = 2$kg $I_{xyz} = 8.33e-4$ $\mu = [1, 0.005, 10^{-4}]$ $l = 0.025m$	bias $= 0.35$ $\sigma = 0.45$

FrankaDrawerOpen	$d_{ee,h} + 0.08 d_{ee,h}^{ang}$
	$-1.0 + d_{drawer}/d_{max}$
		frictionloss = 0.1
		damping=0.1
			bias $= 0.1$
			$\sigma = 10.0$



(a) PendulumSwingup (b) BallInCupSparse (c) FetchPushBlock (d) FrankaDrawerOpen

Figure 1: Comparison of MPQ against baselines during training. The number following H in the legend corresponds to MPC horizon. The dashed grey line shows performance of MPPI with access to true dynamics and no terminal Q function, denoting the upper limit on the performance MPPI can achieve. The horizon for MPPI and MPQ were selected after a rough grid search.

layer of complexity and is beyond the scope of this work. Note that MPQ is a complementary approach to model learning and one can benefit from the other. We make the following observations:

O 1. MPQ *can truncate the planning horizon leading to computational efficiency over MPPI.*

Fig. 1 shows that MPQ outperforms MPPI with the same horizon after only a few training episodes and ultimately performs better than MPPI with a much longer horizon. This phenonmenon can be attributed to: (1) global information encapsulated in the Q function; (2) hardness of optimizing longer sequences; and (3) compounding model error in longer horizon rollouts [36]. In FetchPushBlock, MPPI with a short horizon (H=10) is unable to reach close to the block whereas MPQ with H=10 is able to outperform MPPI with H=64 within the first 30 episodes of training i.e. roughly 2 minutes of interaction with true simulation parameters. In the high-dimensional FrankaDrawerOpen, MPQ with H=10 achieves a success rate of >5 times MPPI with H=10, and outperforms MPPI with H=64 within a few minutes of interaction.

O 2. MPQ *mitigates effects of model-bias through a combination of MPC, entropy regularization and a Q function learned from true system.*

Fig. 1 shows that MPQ with short horizon achieves performance close to, or better than, MPPI with access to true dynamics and a longer horizon (dashed gray line) in all tasks.

O 3. *Using MPC provides stable Q targets leading to sample efficiency over* SoftQLearning

In BallInCupSparse, FetchPushBlock and FrankaDrawerOpen, SoftQLearning is does not converge to a consistent policy whereas MPQ achieves good performance within few minutes of interaction with true system parameters.

Case Study: Learning Policies with Inaccurate Models: Domain Randomization (DR) aims to make a policy learned in simulation robust by randomizing the simulation parameters. However, such policies can be suboptimal with respect to the true parameters due to bias in the randomization distribution.

Q 1. *Can a Q-function learned using rollouts on a real system overcome model bias and perform better than DR?*

We compare MPQ against a DR approach inspired by Peng et al. [17] where simulated rollouts are generated by sampling different parameters at every timestep from a broad distribution shown in Table 1 whereas real system rollouts use the true parameters. Table 2 demonstrates that a Q function learned using DR in simulation is unable to generalize to the true parameters during testing and MPQ has over twice the success rate in BallInCupSparse and thrice in FrankaDrawerOpen.

6 Discussion

In this work we have presented a theoretical connection between information theoretic MPC and entropy-regularized RL that naturally provides an algorithm to leverage the benefits of both. The theoretical insight not only ties together the different fields, but opens avenues to designing pragmatic RL algorithms for real-world systems. While the approach is effective on a range of tasks, some important questions are yet to be answered. First, the optimal horizon for MPC is inextricably tied with the model error and optimization artifacts. Investigating this dependence in a principled manner is important for real-world applications. Another interesting avenue of research is characterizing the performance of a parameterized Q function and using it to adapt the horizon of MPC rollouts for smarter exploration.

Table 2: Average success when training using real system rollouts (ending with REAL) and DR. Test episodes=100. H is horizon of MPC in both training and testing, with H = 1 being SoftQLearning

Task	**Agent**	**Avg. success rate**
BallInCupSparse (350 training episodes)	MPQH4REAL	0.85
	MPQH4DR	0.41
	MPQH1REAL	0.09
	MPQH1DR	0.06
FrankaDrawerOpen (200 training episodes)	MPQH10REAL	0.53
	MPQH10DR	0.17
	MPQH1REAL	0.0
	MPQH1DR	0.0

References

[1] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, pp. 5360–5370, 2017.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979. IEEE, 2019.

[4] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pp. 4754–4765, 2018.

[5] Vishnu R Desaraju and Nathan Michael. Fast nonlinear model predictive control via partial enumeration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1243–1248. IEEE, 2016.

[6] Tom Erez, Kendall Lowrey, Yuval Tassa, Vikash Kumar, Svetoslav Kolev, and Emanuel Todorov. An integrated system for real-time model predictive control of humanoid robots. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 292–299. IEEE, 2013.

[7] Roy Fox, Ari Pakman, and Naftali Tishby. Taming the noise in reinforcement learning via soft updates. *arXiv preprint arXiv:1512.08562*, 2015.

[8] Justin Fu, Sergey Levine, and Pieter Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4019–4026. IEEE, 2016.

[9] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1352–1361. JMLR. org, 2017.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Vikash Kumar, Yuval Tassa, Tom Erez, and Emanuel Todorov. Real-time behaviour synthesis for dynamic hand-manipulation. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6808–6815. IEEE, 2014.

[13] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[14] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[15] Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv preprint arXiv:1811.01848*, 2018.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[17] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE, 2018.

[18] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

[19] Ugo Rosolia and Francesco Borrelli. Learning model predictive control for iterative tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 63(7):1883–1896, 2017.

[20] Stephane Ross and J Andrew Bagnell. Agnostic system identification for model-based reinforcement learning. *arXiv preprint arXiv:1203.1007*, 2012.

[21] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[22] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

[23] John Schulman, Xi Chen, and Pieter Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

[24] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017. URL https://arxiv.org/abs/1705.05065.

[25] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International Conference on Machine Learning*, pp. 5779–5788, 2019.

[26] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529 (7587):484, 2016.

[27] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[28] Wen Sun, J Andrew Bagnell, and Byron Boots. Truncated horizon policy search: Combining reinforcement learning & imitation learning. *arXiv preprint arXiv:1805.11240*, 2018.

[29] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan—episodic mpc improvement. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 336–343. IEEE, 2017.

[30] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind control suite. Technical report, DeepMind, January 2018. URL https://arxiv.org/abs/1801.00690.

[31] Evangelos A Theodorou and Emanuel Todorov. Relative entropy and free energy dualities: Connections to path integral and kl control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 1466–1473. IEEE, 2012.

[32] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30. IEEE, 2017.

[33] Emanuel Todorov. Efficient computation of optimal actions. *Proceedings of the national academy of sciences*, 106(28):11478–11483, 2009.

[34] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

[35] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[36] Arun Venkatraman, Martial Hebert, and J Andrew Bagnell. Improving multi-step prediction of learned time series models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[37] Nolan Wagener, Ching-An Cheng, Jacob Sacks, and Byron Boots. An online learning approach to model predictive control. *arXiv preprint arXiv:1902.08967*, 2019.

[38] 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)*, pp. 1714–1721. IEEE, 2017.

[39] Mingyuan Zhong, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. Value function approximation and model predictive control. In *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, pp. 100–107. IEEE, 2013.

[40] Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. 2008.

A Appendix

A.1 Further experimental details

The learned Q function takes as input the current action and a observation vector per task:

1. PendulumSwingup: $\left[\texttt{cos}(\Theta), \texttt{sin}(\Theta), \dot{\Theta}\right]$ (3 dim)
2. BallInCupSparse: $[x_{ball}, x_{target}, x_b\dot{a}ll, x_t\dot{a}rget, x_{target} - x_{ball}, \texttt{cos}(\Theta), \texttt{sin}(\Theta)]$ (12 dim) where Θ is angle of line joining ball and target.
3. FetchPushBlock: $[x_{gripper}, x_{obj}, x_{obj} - x_{grip}$, gripper opening, $\text{rot}_{obj}, \dot{x}_{obj}, \omega_{obj}$ gripper opening vel, $\dot{x}_{gripper}, d(gripper, obj), x_{goal} - x_{obj}, d(goal, obj), x_{goal}]$ (33 dim)
4. FrankaDrawerOpen: $[x_{ee}, x_h, x_h - x_{ee}, \dot{x}_{ee}, \dot{x}_h, \text{quat}_{ee}, \text{quat}_h, \text{drawer}_{disp}$ $d(ee, h), d^{quat}(ee, h), d^{ang}_{ee,h}]$ (39 dim)

For all our experiments we parameterize Q functions with feedforward neural networks with two layers containing 100 units each and `tanh` activation. We use Adam [11] optimization with a learning rate of 0.001. For generating value function targets in Eq. (22), we use 3 iterations of MPPI optimization except FrankaDrawerOpen where we use 1. The MPPI parameters used are listed in Table 2.

Table 2: Cost function and MPPI parameters

Environment	**Cost function**	**Samples**	Σ	λ	α	γ
PendulumSwingup	$\Theta^2 + 0.1\dot{\Theta}^2$	24	4.0	0.15	0.5	0.9
BallInCupSparse	0 if ball in cup 1 else	36	4.0	0.15	0.55	0.9
FetchPushBlock	$d_{block,goal}$	36	3.0	0.01	0.5	0.9
FrankaDrawerOpen	$d_{ee,h} + 0.08d^{ang}_{ee,h}$ $-1.0 + d_{drawer}/d_{max}$	36	4.0	0.05	0.55	0.9