Enhancing Task Performance of Learned Simplified Models via Reinforcement Learning

Hien Bui[1] and Michael Posa[1]

Abstract—In contact-rich tasks, the hybrid, multi-modal nature of contact dynamics poses great challenges in model representation, planning, and control. Recent efforts have attempted to address these challenges via data-driven methods, learning dynamical models in combination with model predictive control. Those methods, while effective, rely solely on minimizing forward prediction errors to hope for better task performance with MPC controllers. This weak correlation can result in data inefficiency as well as limitations to overall performance. In response, we propose a novel strategy: using a policy gradient algorithm to find a simplified dynamics model that explicitly maximizes task performance. Specifically, we parameterize the stochastic policy as the perturbed output of the MPC controller, thus, the learned model representation can directly associate with the policy or task performance. We apply the proposed method to contact-rich tasks where a three-fingered robotic hand manipulates previously unknown objects. Our method significantly enhances task success rate by up to 15% in manipulating diverse objects compared to the existing method while sustaining data efficiency. Our method can solve some tasks with success rates of 70% or higher using under 30 minutes of data. All videos and codes are available at https://sites.google.com/view/lcs-rl.

I. INTRODUCTION

In many robotics tasks, such as dexterous manipulation and locomotion, robots frequently need to make and break contact with the environment. Yet, finding explicit models and policies that can exploit the hybrid complex interaction of the robot with its environment to solve the tasks remains a challenge. Some works in model-based control [1]–[4] have attempted to explicitly identify contact modes and plan the contact sequences. However, these approaches face scalability challenges as the number of contact modes increases.

Recent data-driven methods have made significant advances in tackling that scalability issue, broadly offering two primary strategies. Modern model-free reinforcement learning (RL) directly parameterizes control policies with deep neural networks, then iteratively improves the policies through large-scale trial and error [5]–[7]. However, because of their data inefficiency, carrying out experiments on real robotic systems is always resource-intensive and time-consuming. In contrast, a line of work, often referred to as model-based RL [8]–[14], leverages the expressiveness power of deep neural networks to learn intricate dynamic models. The learned models are subsequently employed for trajectory planning via random shooting techniques. Despite improving data efficiency compared to model-free RL, model-based RL remains data-intensive because conventional
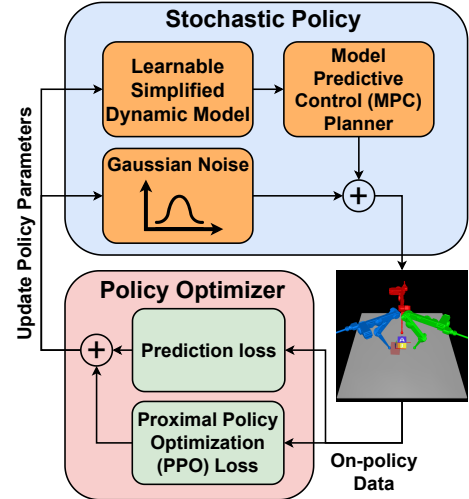

Fig. 1: The diagram demonstrates our proposed framework of learning simplified dynamic models for solving contact-rich manipulation tasks in low data regimes. Our framework proposes an iterative learning loop that consists of main components: a stochastic policy and a policy optimizer. **Top panel**: Using learned dynamic models under MPC scheme and Gaussian noise to construct the stochastic policy. **Bottom panel:** Combining PPO and prediction loss to optimize the policy parameters with the collected on-policy data.

methods for learning deep dynamic models struggle to capture stiffness and multi-modal contact dynamics [15], [16]. Moreover, by adopting simpler model representations such as time-varying linear or Gaussian Process models, some works [17]–[19] demonstrate good performance with limited data.

The most recent work [20] shows great performance with only a few minutes of data by learning a simple multi-contact dynamic model, or reduced-order model, and combining it with an MPC planner. However, two main building blocks of this work, the dynamic model fitting and planning with the learned model, appear as two de-coupled optimization problems repeated over many cycles of learning. More specifically, ensuring better forward prediction capability of the learned dynamic model is necessary but might not be sufficient to achieve better task performance, leading to limitations of data efficiency and task performance. This issue is known as *objective mismatch* [21].

This paper presents LCS-RL, a low-dimensional RL framework, to address the above issue, aiming to further enhance task performance and data efficiency. Particularly, the proposed framework directly bridges the dynamic model learning part to task performance optimization via a policy

[1]The authors are with the GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104, USA {xuanhien, posa}@seas.upenn.edu

gradient algorithm.

A. Contributions

1) We present LCS-RL, a novel framework that leverages the combination of RL and simple multi-contact models for solving contact-rich tasks. Specifically, our framework applies a reinforcement learning algorithm to directly maximize the task performance of simplified models in combination with the MPC planner.
2) We show that the proposed method consistently achieves higher task performance, up to 15% in three-fingered robot manipulation tasks with various objects compared to the prior methods [10], [20]. In addition, our method is data efficient as it can solve some dexterous manipulation tasks with 70% to 96% success rates using just under 30 minutes of data.
3) We also demonstrate that the learned LCS model of one object can be transferred to other objects, drastically improving data efficiency.

II. RELATED WORK

A. Differentiable MPC and Reinforcement Learning

The prior works [22]–[24] are the most relevant to ours: forming parametric MPC with learnable either cost functions or dynamic constraints, and optimizing its parameters via some losses such as imitation or RL loss. However, those works express some limitations. Amos et al. [23] propose to use simple linear dynamic models and limit their experiments to imitation learning on low-dimensional tasks. Meanwhile, Esfahani et al. [24] focus more on robust MPC, optimizing parameters of the cost functions for better performance with the presence of disturbances and uncertainties, but critically require access to ground-truth dynamics.

B. Baseline for Comparisons

Jin et al. [20] suggest that there exists a simple model that can adequately capture task-relevant contact dynamics, thereby enabling both high performance and real-time control for contact-rich manipulation. In particular, the authors propose to use a reduced-order hybrid model to represent and use the model predictive controller for planning. Since the model is far simpler than deep neural networks, much less data is required for model learning. Their framework achieves high task performance with under 5 minutes of data. In this paper, we compare the task performance and data efficiency of our proposed method against this baseline in some dexterous manipulation tasks.

III. BACKGROUNDS

A. Linear Complementarity Systems

A discrete-time linear complementarity system (LCS) is a piecewise-affine system, where the state evolution is governed by linear dynamics in (1a) and a linear complementarity problem (LCP) in (1b).

$$x_{t+1} = Ax_t + Bu_t + C\lambda_t + d, \tag{1a}$$
$$0 \leq \lambda_t \perp Dx_t + Eu_t + F\lambda_t + c \geq 0. \tag{1b}$$

Here, $x_t \in \mathbb{R}^{n_x}$, $u_t \in \mathbb{R}^{n_u}$, and $\lambda_t \in \mathbb{R}^{n_\lambda}$ are respectively the system state, action, and the complementarity variable at time step t. And, $x_{t+1} \in \mathbb{R}^{n_x}$ is the system state at the next time step $t + 1$. Moreover, the matrix $A \in \mathbb{R}^{n_x \times n_x}$ defines the autonomous dynamics and matrix $B \in \mathbb{R}^{n_x \times n_u}$ captures the effect of actions on states. And, the matrix $C \in \mathbb{R}^{n_x \times n_\lambda}$ and $d \in \mathbb{R}^{n_x}$ describe the effect of the contact forces and the constant forces acting on the state respectively. Other matrices $D \in \mathbb{R}^{n_\lambda \times n_x}, E \in \mathbb{R}^{n_\lambda \times n_a}, F \in \mathbb{R}^{n_\lambda \times n_\lambda}$ and $c \in \mathbb{R}^{n_\lambda}$ altogether capture the relationship between states, actions, and contact forces.

B. Learning Linear Complementarity Systems

Given a data buffer $\mathcal{D}$ that contains some state transitions (x_t, u_t, x_{t+1}), we can learn all matrix and vector parameters of an LCS model (A, B, C, d, D, E, F, c) in (1) by using the gradient descent method with the violation-based loss, proposed by Jin et al. [25]

$$\mathcal{L}_{\text{vio}}^{\Theta} = \min_{\lambda_t \geq 0, \phi_t \geq 0} \frac{1}{2} \|Ax_t + Bu_t + C\lambda_t + d - x_{t+1}\|^2$$
$$+ \frac{1}{\xi} \left(\lambda_t^T \phi_t + \frac{1}{2\gamma} \|Dx_t + Eu_t + F\lambda_t + c - \phi_t\|^2 \right). \tag{2}$$

Particularly, the loss $\mathcal{L}_{\text{vio}}^{\Theta}$ itself is an optimization problem whose first and second terms specify the violation of the affine dynamics (1a) and the LCP constraint (1b), respectively. Under the condition $0 < \gamma \leq \sigma_{min}(F + F^T)$, finding λ_t to minimize the second term is equivalent to directly solving an LCP in (1b) for λ_t, but poses a better-conditioned landscape for $\mathcal{L}_{\text{vio}}^{\Theta}$, thus enabling the identification of multi-modal and stiff dynamics. The hyper-parameter $\xi > 0$ aims to balance two terms of the loss $\mathcal{L}_{\text{vio}}^{\Theta}$; and $\phi_t \in \mathbb{R}^{n_\lambda}$ is an introduced slack variable for the complementarity equation. Full explanations of the loss formulation and its hyper-parameters can be found in [25].

As proven in [25], using Envelope Theorem [26], we can analytically compute the gradient of the violation-based loss with respect to LCS parameters $\frac{d\mathcal{L}_{vio}^{\Theta}}{d\Theta}$ without differentiating through the solution of the optimization problem.

C. Model Predictive Controller with LCS

Utilizing LCS to represent the dynamics model, one can construct a model predictive controller (MPC) as follows:

$$\min_{u_t, u_{t+1}, \ldots, u_{t+H-1}} \sum_{k=t}^{t+H-1} \mathcal{C}(x_k, u_k) + \mathcal{C}_f(x_{t+H})$$
$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k + C\lambda_k + d,$$
$$0 \leq \lambda_k \perp Dx_k + Eu_k + F\lambda_k + c \geq 0,$$
$$u_{\min} \leq u_k \leq u_{\max}. \tag{3}$$

where H is the planning horizon; $\mathcal{C}$ and $\mathcal{C}_f$ are the path and terminal cost functions. And, u_{min} and u_{max} are the lower and upper bounds of actions.

Given any initial state x_t, we solve the LCS-MPC in (3) to plan a sequence of optimal actions $[u_t, u_{t+1}, \ldots, u_{t+H-1}]$ that minimizes the total cost, then select the first action u_t to

Algorithm 1: Warm-up phase for optimizing LCS-MPC Policy

Parameterization: LCS-MPC policy parameters $\boldsymbol{\theta}$;
Hyper-parameters: The number of warm-up iterations M; the number of policy improvement steps N_p; and learning rate η;
Initialization: $\boldsymbol{\theta}_0$, ϕ_0, empty data buffer $\mathcal{D}$;
for $k = 0, 1, ..., M$ **do**
    Collect N rollout trajectories by running the LCS-MPC stochastic policy $\pi_{\boldsymbol{\theta}_k}$ and add to $\mathcal{D}$
    for $i \leftarrow 0$ **to** N_p **do**
        Using data in $\mathcal{D}$, compute the gradient $\frac{d\mathcal{L}_{vio}^{\boldsymbol{\theta}_i}}{d\boldsymbol{\theta}_i}$
        Update $\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i - \eta \frac{d\mathcal{L}_{vio}^{\boldsymbol{\theta}_i}}{d\boldsymbol{\theta}_i}$
    end
end
Save the final parameters $\boldsymbol{\theta}_M$ for the main phase

---

apply on the robot and repeat the process in every time step in a receding horizon manner. To efficiently solve the LCS-MPC, we employ the direct trajectory optimization method [27]. This approach simultaneously searches over trajectories of $\boldsymbol{x}_{0:T}$, $\boldsymbol{u}_{0:T-1}$, and $\boldsymbol{\lambda}_{0:T-1}$, treating the LCS dynamics as a separate constraint for each time step. Also, we use the IPOPT solver [28] to solve such nonlinear problems.

D. Proximal Policy Optimization

Proximal Policy Optimization (PPO) [5] is a policy gradient algorithm that focuses on determining how to make the most significant policy improvement using current data, all while avoiding excessive steps that could lead to performance collapse. Particularly, the PPO loss is defined as follows:

$$\mathcal{L}_{PPO}^{\boldsymbol{\theta}} = -\frac{1}{|\mathcal{D}|T} \sum_{\tau \in D} \sum_{t=0}^{T} \begin{cases} \max\left(h_t^{\theta}, 1 - \epsilon\right) A_t & \text{if } A_t < 0 \\ \min\left(h_t^{\theta}, 1 + \epsilon\right) A_t & \text{if } A_t \geq 0, \end{cases}$$ (4)

where $\mathcal{D}$ and $|\mathcal{D}|$ are the data buffer and its size, that buffer consists of on-policy rollout trajectories τ, and T is the length of trajectories. There are two key quantities in (4): the ratio $h_t^{\boldsymbol{\theta}} = \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{u}_t|\boldsymbol{x}_t)}$ and the advantage function A_t. Here, the ratio $h_t^{\boldsymbol{\theta}}$ indicates how much the new policy differs from the old one. The scalar ϵ defines the bounds of $h_t^{\boldsymbol{\theta}}$, which are often referred to as the trust region of policy improvements. In addition, the advantage function A_t guides the policy search by measuring whether a certain action is a good or bad decision within a given state

$$A_t = r_t + \gamma \mathcal{V}_{\phi}(\boldsymbol{x}_{t+1}) - \mathcal{V}_{\phi}(\boldsymbol{x}_t),$$ (5)

where r_t is the reward obtained by executing action $\boldsymbol{u}_t$ at state $\boldsymbol{x}_t$ and γ is the discount factor. The value function $\mathcal{V}_{\phi}(\boldsymbol{x}_t)$ and $\mathcal{V}_{\phi}(\boldsymbol{x}_{t+1})$ estimate the expected total rewards if we follow the current policy from state $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$ till the end of trajectories.

Algorithm 2: Main phase for optimizing LCS-MPC Policy using the PPO algorithm

---

Parameterization: LCS-MPC policy parameters $\boldsymbol{\theta}$ and value function $\mathcal{V}_{\phi}$
Hyper-parameters: Total number of iterations K, the number of policy improvement steps $\bar{N}_p$; Learning rate $\bar{\eta}$ for the policy optimization; Loss weighting parameter β in (7); and also the discount factor γ for computing the advantage values.
Initialization: $\boldsymbol{\theta}_M$ obtained from the warm-up phase, ϕ_0, and empty data buffer $\mathcal{D}$
for $k = 0, 1, ..., K$ **do**
    Empty buffer D, collect $\bar{N}$ new trajectories by running the LCS-MPC policy $\pi_{\boldsymbol{\theta}_k}$, and add to $\mathcal{D}$
    For each trajectory, compute the bootstrapped total reward $R_t = r(\boldsymbol{x}_t, \boldsymbol{u}_t) + \gamma \mathcal{V}_{\phi_k}(\boldsymbol{x}_{t+1})$ and the advantage values $A_t = R_t - \mathcal{V}_{\phi_k}(\boldsymbol{x}_t)$ for $t = 0, 1, \ldots, T$
    for $i \leftarrow 0$ **to** $\bar{N}_p$ **do**
        Compute the combined loss gradient $\frac{d\mathcal{L}_c^{\boldsymbol{\theta}_i}}{d\boldsymbol{\theta}_i}$
        Update $\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i - \bar{\eta} \frac{d\mathcal{L}_c^{\boldsymbol{\theta}_i}}{d\boldsymbol{\theta}_i}$
    end
    Fit value function $\mathcal{V}_{\phi}$ by performing regression with mean-square error
    $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \left(\mathcal{V}_{\phi}(\boldsymbol{x}_t) - R_t\right)^2$$
end

IV. PRACTICAL ALGORITHM

In this section, we introduce our framework, LCS-RL, that utilizes a reinforcement learning algorithm, here we use PPO, to optimize an LCS dynamic model (in combination with model predictive control) for solving contact-rich tasks.

First, we formulate a stochastic policy, called the LCS-MPC stochastic policy, by adding Gaussian noise to the output of the LCS-MPC planner in (3). In other words, the LCS-MPC policy is directly parameterized by the LCS model. Then, we use the combination of the PPO loss and the violation-based loss given in (7) to improve the task performance of that policy.

We address the poor data efficiency of the PPO algorithm by leveraging data-efficient model learning at the start and then transitioning to PPO when in a good neighborhood. Therefore, our framework consists of two phases: the warm-up phase and the main phase. In the warm-up phase, we follow the algorithm proposed in [20], solely employing the violation-based loss (2) to quickly learn the parameters of the LCS model that can achieve good task performance. Subsequently, we use the learned LCS model to accelerate the main phase in which we start using the PPO algorithm. In practice, we find that having the warm-up phase offers more stable and progressive training than involving PPO right from the beginning. Theoretically, we could merge two phases

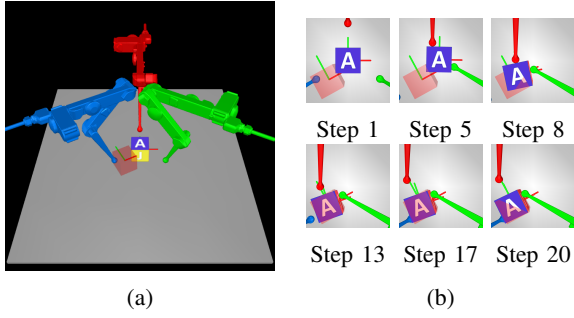(a)   (b)

Fig. 2: TriFinger dexterous manipulation tasks. (a) shows the simulation environment that is constructed using MuJoCo physics engine [29]. In this task, the three fingers need to push the cube towards a random target pose, visualized by the red transparent cube. (b) is an example of a rollout trajectory that demonstrates how the fingers approach, make, and break contacts to reposition the cube.

together and switch the loss upon transition. However, each phase requires a different set of hyper-parameters such as the learning rate, the gradient descent optimizer, or the number of updating iterations, so it is simpler to keep two phases separated. Details of the warm-up phase and main phase are given in Algorithm 1 and 2.

A. LCS-MPC Stochastic Policy

The LCS-MPC policy is the probability density of an action distribution associated with the current state $\boldsymbol{x}_t$:

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t) = \frac{\exp\left(-\frac{1}{2}(\boldsymbol{u}_t - \mu_{\boldsymbol{\Theta}}(\boldsymbol{x}_t))^T \Sigma^{-1}(\boldsymbol{u}_t - \mu_{\boldsymbol{\Theta}}(\boldsymbol{x}_t))\right)}{(2\pi)^{n_u/2} \det(\Sigma)^{1/2}}, \tag{6}$$

where $\mu_{\boldsymbol{\Theta}} \in \mathbb{R}^{n_u}$ is the deterministic action or the optimal output of the LCS-MPC planner, and $\Sigma \in \mathbb{R}^{n_u \times n_u}$ is the covariance matrix that indicates the noise magnitude. Here, $\boldsymbol{\Theta} = (A, B, C, \boldsymbol{d}, D, E, F, \boldsymbol{c})$ is actually the LCS parameters. And, $\boldsymbol{\theta} = [\boldsymbol{\Theta}, \Sigma]$ denotes the joint vector of the policy's learnable parameters. The added noise encourages exploration, thus avoiding low-quality local minima. Usually, the noise magnitude is large at the beginning, then gradually decreases as the task performance would be higher if the policy exploits the knowledge it has acquired.

B. Loss for Optimizing LCS Model

Throughout our framework, we employ two types of loss: the violation-based loss and the PPO loss. The former focuses on improving the forward prediction capability of the LCS model while the latter focuses on directly enhancing the task performance of the LCS-MPC planner. By introducing a hyper-parameter $\beta \in [0, 1]$, we could balance the contributions of the PPO loss and the violation-based loss, resulting in the combined loss:

$$\mathcal{L}_c^{\boldsymbol{\theta}} = \beta \mathcal{L}_{PPO}^{\boldsymbol{\theta}} + (1 - \beta)\mathcal{L}_{vio}^{\boldsymbol{\theta}}, \tag{7}$$

In order to optimize the parameters of LCS-MPC stochastic policy, one must compute the gradient of the PPO loss and the violation-based loss with respect to the policy parameters. In section III-B we have already mentioned the method for
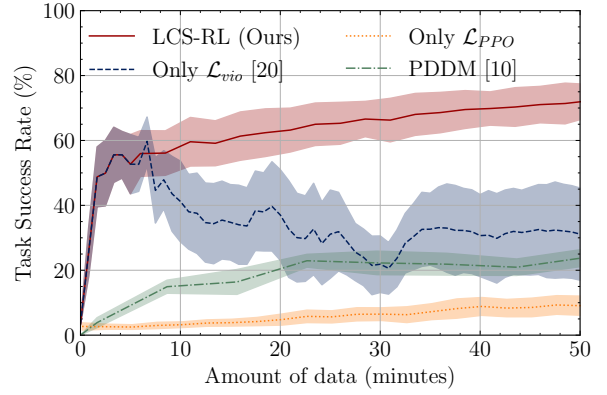

Fig. 3: Learning curves of the TriFinger Moving Cube task. The red, blue, orange, and green lines show the average task success rate of our proposed method, the prior method [20], a method that uses PPO without a warm-up phase, and PDDM [10] respectively. At the beginning of the training, our method and the prior method [20] share the same performance since the same algorithm is used. However, the transition occurs after collecting 6 minutes of data, when our method switches to fully employ the PPO algorithm. Shaded regions indicate normal t-score 95% confidence intervals.

calculating $\frac{d\mathcal{L}_{vio}^{\boldsymbol{\theta}}}{d\boldsymbol{\theta}}$. Moving forward, we will illustrate the process for computing $\frac{d\mathcal{L}_{PPO}^{\boldsymbol{\theta}}}{d\boldsymbol{\theta}}$.

C. Gradient of PPO Loss

The original PPO algorithm parameterizes policies via deep neural networks [5], thus computing the gradient of the PPO loss over policy parameters can be simply done via automatic differentiation. However, it does not apply to our case since our policy is actually an MPC planner. Differentiating through the MPC requires special treatment.

Given the explicit form of PPO loss in (4), using chain rule, the gradient with respect to $\boldsymbol{\theta}$ can be computed as follows:

$$\frac{d\mathcal{L}_{PPO}^{\boldsymbol{\theta}}}{d\boldsymbol{\theta}} = \frac{1}{|\mathcal{D}|T} \sum_{\tau \in D} \sum_{t=0}^{T} \begin{cases} \frac{dh_t^{\boldsymbol{\theta}}}{d\pi_{\boldsymbol{\theta}}} \frac{d\pi_{\boldsymbol{\theta}}}{d\boldsymbol{\theta}} A_t & \text{if } h_t^{\boldsymbol{\theta}} \geq 1 - \epsilon; \ A_t < 0 \\ 0 & \text{if } h_t^{\boldsymbol{\theta}} < 1 - \epsilon; \ A_t < 0 \\ \frac{dh_t^{\boldsymbol{\theta}}}{d\pi_{\boldsymbol{\theta}}} \frac{d\pi_{\boldsymbol{\theta}}}{d\boldsymbol{\theta}} A_t & \text{if } h_t^{\boldsymbol{\theta}} \leq 1 + \epsilon; \ A_t \geq 0 \\ 0 & \text{if } h_t^{\boldsymbol{\theta}} > 1 + \epsilon; \ A_t \geq 0. \end{cases} \tag{8}$$

Due to the clipping effect of the PPO loss, the gradients are zero when the improvement steps of the PPO policy $h_t^{\boldsymbol{\theta}}$ are outside of trust region $[1 - \epsilon, 1 + \epsilon]$. Hence, we are left to compute the gradients if $h_t^{\boldsymbol{\theta}}$ stays within the trust region. To compute (8), one must compute $\frac{dh_t^{\boldsymbol{\theta}}}{d\pi_{\boldsymbol{\theta}}}$ and $\frac{d\pi_{\boldsymbol{\theta}}}{d\boldsymbol{\theta}}$. While it is straightforward to evaluate $\frac{dh_t^{\boldsymbol{\theta}}}{d\pi_{\boldsymbol{\theta}}}$, computing $\frac{d\pi_{\boldsymbol{\theta}}}{d\boldsymbol{\theta}}$ requires differentiation of the optimal actions of the MPC $\mu_{\boldsymbol{\Theta}}(\boldsymbol{x}_t)$ with respect to its parameters $\boldsymbol{\Theta}$. We compute this derivative via perturbations of Karush–Kuhn–Tucker (KKT) conditions [30], with details given in [31]. Also, note that the MPC problem is not always classically differentiable (e.g. when strict complementarity does not hold in the KKT conditions), but we have not found this problematic in practice.

V. Experiments and Results

In this section, we will verify our proposed framework on the three-fingered robotic hand manipulation task that was first proposed by [20] (see Fig. 2). We call it the TriFinger Moving Cube task. In the first experiment, we show a comparison of the task performance of the LCS model trained by our method and prior methods. Next, we replace the cube with other objects that have more complex shapes and repeat the same experiment. Lastly, we demonstrate that the data efficiency of our framework can be greatly improved via transfer learning. In order to guarantee statistically meaningful results, for each experiment, we have 10 runs with 10 random seeds. Also, we compute the task success rate by evaluating the learned models with 1000 random goal poses and aggregate results. All videos and codes are available at https://sites.google.com/view/lcs-rl.

A. TriFinger Moving Cube Task

In this task, a TriFinger robot attempts to align a 6 cm-sized cube with random goal poses on a planar surface. Each episode consists of up to 20 steps, where each step takes 0.1 seconds.

1) States and Actions: We define the system state

$$\boldsymbol{x} = [\boldsymbol{p}_{cube}, \ \alpha_{cube}, \ \boldsymbol{p}_{fingertips}] \in \mathbb{R}^9, \tag{9}$$

where $\boldsymbol{p}_{cube} \in \mathbb{R}^2$ is the xy position of the cube; $\alpha_{cube} \in \mathbb{R}$ is the rotation angle around the z (vertical) axis; and $\boldsymbol{p}_{fingertips} \in \mathbb{R}^6$ are the xy positions of three fingertips. We define the actions as deviations from the current positions of three fingertips in the Cartesian space. In addition, we impose safety limits on actions (element-wise) to constrain how far fingertips can move in one time step.

$$\boldsymbol{u} = \Delta \boldsymbol{p}_{fingertips} \in \mathbb{R}^6, \\ u_i \in [-0.015, \ 0.015] \ \text{m}. \tag{10}$$

We employ operational space control (OSC) [32] in the lower-level controller to map action $\boldsymbol{u}$ to the joint torque of each finger. We also utilize the OSC controller to maintain fingertips at a constant height as this TriFinger task involves only planar manipulation.

2) Task Space: We use the same bounds for task space as in [20], from which the goal poses are uniformly sampled

$$[-0.06, \ 0.06]^T \ \text{m} \leq \boldsymbol{p}_{goal} \leq [0.06, \ 0.06]^T \ \text{m}, \\ -0.5 \ \text{rads} \leq \alpha_{goal} \leq 0.5 \ \text{rads}. \tag{11}$$

3) Task Success Criteria: When the cube pose is near the goal pose and within some tolerances, we can consider that the task is successfully completed. The goal tolerance values are selected to establish the right level of difficulty as too stringent tolerances make the task impossible to solve. We follow some previous works on TriFinger tasks [33], [34] to set the tolerances as follows:

$$\|\boldsymbol{p}_{cube} - \boldsymbol{p}_{goal}\| \leq 0.02 \ \text{m}, \\ \|\alpha_{cube} - \alpha_{goal}\| \leq 0.2 \ \text{rads}, \tag{12}$$

where $\boldsymbol{p}_{cube}$ and α_{cube}^T are the xy position and orientation of the cube at the last time step T; $\boldsymbol{p}_{goal} \in \mathbb{R}^2$ and $\alpha_{goal} \in \mathbb{R}$ together specify the goal pose.

Object	Peak Success Rate (%)		Final Success Rate (%)	
	Only $\mathcal{L}_{\mathbf{vio}}$	LCS-RL (Ours)	Only $\mathcal{L}_{\mathbf{vio}}$	LCS-RL (Ours)
Sugar Box	87.5 ± 9.2	**95.9 ± 2.4**	44.3 ± 26.2	**95.9 ± 2.4**
Fish Can	59.1 ± 7.3	**69.9 ± 8.1**	38.8 ± 13.1	**69.9 ± 8.1**
Mug	32.5 ± 8.4	**44.6 ± 3.9**	10.0 ± 6.3	**44.6 ± 3.9**
Wrench	45.5 ± 8.7	**60.0 ± 6.5**	11.5 ± 8.6	**59.5 ± 6.6**
Clamp	24.7 ± 5.3	**39.3 ± 9.7**	6.1 ± 5.6	**38.7 ± 10.2**
Banana	28.4 ± 5.8	**35.5 ± 5.3**	7.5 ± 6.6	**35.5 ± 5.3**

TABLE I: Comparison task success rates between our method [20] and prior method on diverse objects.

4) Cost function for the LCS-MPC: We utilize the same cost function for the LCS-MPC as in the prior work [20]:

$$\mathcal{J} = \sum_{k=t}^{t+H-1} \mathcal{C}(\boldsymbol{x}_k, \boldsymbol{u}_k) + \mathcal{H}(\boldsymbol{x}_{t+H}),$$

$$\mathcal{C} = 10.0 \left\| \boldsymbol{p}_{\text{fingertips}} - \boldsymbol{p}_{\text{cube}} \right\|^2 + 200.0 \left\| \boldsymbol{p}_{\text{cube}} - \boldsymbol{p}^{\text{goal}} \right\|^2 \\ + 0.3 \left(\alpha_{\text{cube}} - \alpha_{\text{goal}} \right)^2 + 200 \|\boldsymbol{u}\|^2,$$

$$\mathcal{H} = 6.0 \left\| \boldsymbol{p}_{\text{fingertips}} - \boldsymbol{p}_{\text{cube}} \right\|^2 + 200.0 \left\| \boldsymbol{p}_{\text{cube}} - \boldsymbol{p}_{\text{goal}} \right\|^2 \\ + 1.5 \left(\alpha_{\text{cube}} - \alpha^{\text{goal}} \right)^2, \tag{13}$$

5) Reward function for PPO: We employ both dense and sparse reward functions for the PPO algorithm. The dense reward function $r_t(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is simply the negation of the cost function $\mathcal{C}$ in (13). One particular reason for this choice of reward function is to ensure both PPO and the MPC planner of the stochastic policy align in the same direction toward task completion.

At the end of the rollout trajectory, we add a negative sparse reward to penalize for not completing the task:

$$r(\boldsymbol{x}_{T-1}, \boldsymbol{u}_{T-1}) = -10.0 \times (1 - is\_task\_completed), \tag{14}$$

In practice, we find that sparse reward helps to accelerate the training significantly.

B. Results of the TriFinger Moving Cube Task

To demonstrate the effectiveness of LCS-RL, we compare against the prior method [20], which trains purely on $\mathcal{L}_{vio}^{\boldsymbol{\theta}}$, against PPO without a warm-up phase. We also compare against a state-of-the-art model-based RL approach PDDM [10] to demonstrate the utility of simple, non-smooth models over deep neural networks for manipulation. Note that in the main phase of our method, we set $\beta = 1.0$ for the combined loss in (7), meaning that only the PPO loss is used. The results are shown in Fig.3.

When utilizing only the violation-based loss, the mean success rate peaks at 55% after 7 minutes of data, then fluctuates and decreases as more data is collected, and finally stops at approximately 30%. In contrast, starting with the same performance at 6 minutes of data, our method improves the task performance throughout the training, reaching 65% of success rate after 25 minutes of data and 71.4% at the end of the training. Since the LCS models have limited expressiveness power, even if we optimize LCS models for better capability of forward prediction, this capability might not be optimally assigned to regions of state space
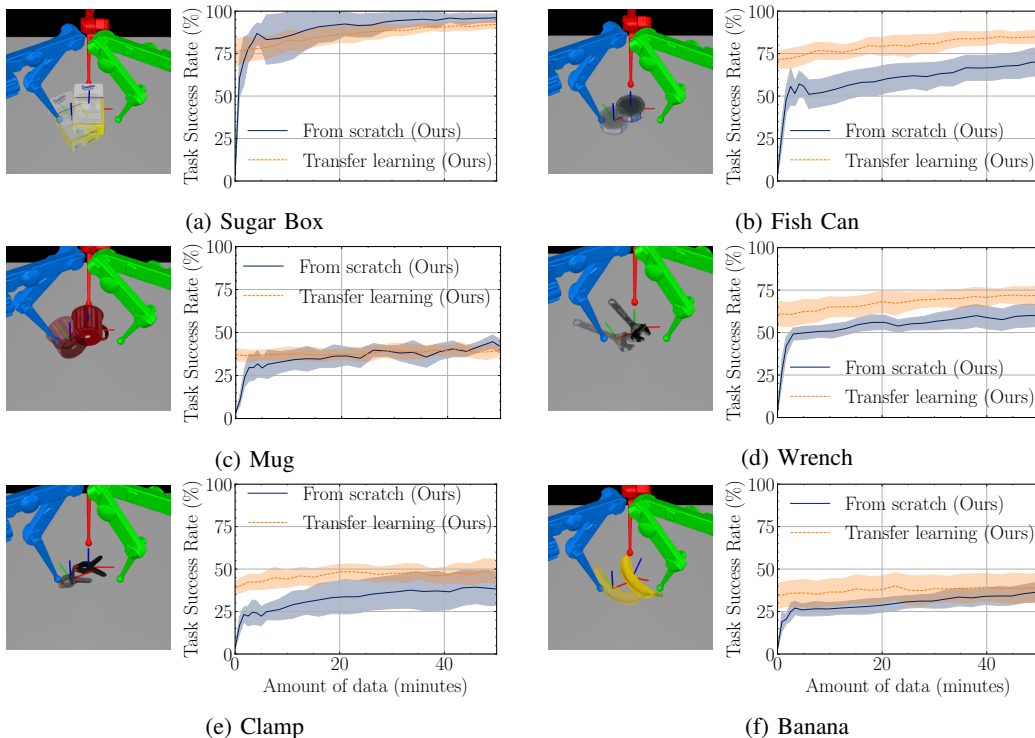
Fig. 4: Comparison task performance between learning the LCS model from scratch and pre-trained models (obtained from training with the TriFinger Moving Cube task) on the YCB objects using our LCS-RL framework.

where accuracy is needed for task performance. As a result, the overall task performance might drop significantly. Our method does not suffer from that issue because the only objective of PPO is encouraging the policy, to repeat good trajectories and avoid bad trajectories.

In addition, the PPO-only method and PDDM [10] have the lowest task performances throughout the training, achieving merely 10% and 20% for the final task success rate.

C. TriFinger Manipulating Diverse Objects

We run a set of experiments on the TriFinger Moving Object task, which is similar to the TriFinger Moving Cube task, but the cube is replaced by other objects with non-convex, highly intricate shapes. Those objects, including sugar box, fish can, mug, wrench, clamp, and banana, are selected from the YCB object and model set [35]. As seen from Table I, our method consistently outperforms the prior method in [20] given the same amount of data, gaining from 8% (sugar box) to 15% (clamp) higher task success rate.

D. Transfer Learning

To illustrate the transfer learning capabilities of our LCS-RL framework, we employ the LCS model initially trained on the TriFinger Moving Cube task as the starting point for training on other objects. The results in Fig. 4 show that our LCS-RL framework is highly suitable for transfer learning. Particularly, we can observe that transfer learning significantly accelerates the training, yielding even higher final task success rates in all objects (except for the sugar box), compared to the training from scratch model.

VI. CONCLUSIONS

In conclusion, we present LCS-RL, a novel approach that leverages a reinforcement learning algorithm to directly maximize the task performance of the LCS model in combination with the MPC planner. We demonstrate that the proposed method attains higher task performance and greater sample efficiency compared to prior methods in TriFinger robot tasks involving pushing and rotating various objects. In addition, we show that our method is highly suitable for transfer learning, which further helps to improve data efficiency.

Our framework is not limited to only the PPO algorithm since any RL algorithms can be incorporated. Thus, one direction for future work is to explore other RL algorithms and employ them in our framework. There are off-policy RL algorithms such as TD3 [6] or SAC [7], known for better data efficiency when compared to on-policy algorithms like PPO [36]. Nevertheless, this advantage may not be as evident in situations with limited data.

Lastly, we observe the limitation of using the LCS model to represent system dynamic models, especially when dealing with complex geometries such as bananas or clamps. We would like to explore alternative structured models that are still much simpler than neural networks and also exhibit nonlinear components, though with a trade-off in data efficiency. One potential candidate would be the Nonlinear Complementarity System (NCS) model.

References

[1] T. Marcucci and R. Tedrake, "Warm Start of Mixed-Integer Programs for Model Predictive Control of Hybrid Systems," *IEEE Transactions on Automatic Control*, vol. 66, pp. 2433–2448, 2019.

[2] D. Frick, A. Georghiou, J. L. Jerez, A. Domahidi, and M. Morari, "Low-complexity method for hybrid MPC with local guarantees," *SIAM Journal on Control and Optimization*, vol. 57, no. 4, pp. 2328–2361, 2019.

[3] A. Aydinoglu, A. Wei, and M. Posa, "Consensus Complementarity Control for Multi-Contact MPC," *arXiv preprint arXiv:2304.11259*, 2023.

[4] A. Aydinoglu and M. Posa, "Real-Time Multi-Contact Model Predictive Control via ADMM," in *2022 International Conference on Robotics and Automation (ICRA)*. Philadelphia, PA, USA: IEEE, 2022, pp. 3414–3421.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[6] S. Fujimoto, H. Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018, pp. 1587–1596.

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80. PMLR, 2018, pp. 1861–1870.

[8] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7559–7566.

[9] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[10] A. Nagabandi, K. Konoglie, S. Levine, and V. Kumar, "Deep Dynamics Models for Learning Dexterous Manipulation," in *Conference on Robot Learning*. PMLR, 2020, pp. 1101–1112.

[11] A. S. Morgan, D. Nandha, G. Chalvatzaki, C. D'Eramo, A. M. Dollar, and J. Peters, "Model Predictive Actor-Critic: Accelerating Robot Skill Acquisition with Deep Reinforcement Learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 6672–6678.

[12] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine, "Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control," *arXiv:1812.00568 [cs]*, 2018.

[13] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, and S. Levine, "SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97. PMLR, 2019, pp. 7444–7453.

[14] R. Ghugare, H. Bharadhwaj, B. Eysenbach, S. Levine, and R. Salakhutdinov, "Simplifying Model-based RL: Learning Representations, Latent-space Models, and Policies with One Objective," *arXiv preprint arXiv:2209.08466*, 2022.

[15] M. Parmar, M. Halm, and M. Posa, "Fundamental Challenges in Deep Learning for Stiff Contact Dynamics," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Prague, Czech Republic: IEEE, 2021, pp. 5181–5188.

[16] B. Bianchini, M. Halm, N. Matni, and M. Posa, "Generalization Bounded Implicit Learning of Nearly Discontinuous Functions," in *Proceedings of The 4th Annual Learning for Dynamics and Control Conference (L4DC)*, ser. Proceedings of Machine Learning Research, vol. 168, 2022, pp. 1112–1124.

[17] V. Kumar, E. Todorov, and S. Levine, "Optimal control with learned local models: Application to dexterous manipulation," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. Stockholm, Sweden: IEEE, 2016, pp. 378–383.

[18] S. Levine and V. Koltun, "Guided Policy Search," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28. Atlanta, Georgia, USA: PMLR, 2013, pp. 1–9.

[19] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A Model-Based and Data-Efficient Approach to Policy Search," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. Madison, WI, USA: Omnipress, 2011, pp. 465–472.

[20] W. Jin and M. Posa, "Task-Driven Hybrid Model Reduction for Dexterous Manipulation," *arXiv preprint arXiv:2211.16657*, 2022.

[21] N. Lambert, B. Amos, O. Yadan, and R. Calandra, "Objective Mismatch in Model-based Reinforcement Learning," *arXiv preprint arXiv:2002.04523*, 2021.

[22] M. Okada, L. Rigazio, and T. Aoshima, "Path Integral Networks: End-to-End Differentiable Optimal Control," *arXiv preprint arXiv:1706.09597*, 2017.

[23] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for End-to-end Planning and Control," in *Advances in Neural Information Processing Systems*, 2019.

[24] H. N. Esfahani, A. B. Kordabad, and S. Gros, "Approximate Robust NMPC using Reinforcement Learning," in *2021 European Control Conference (ECC)*, Rotterdam, Netherlands, 2021.

[25] W. Jin, A. Aydinoglu, M. Halm, and M. Posa, "Learning Linear Complementarity Systems," in *Proceedings of The 4th Annual Learning for Dynamics and Control Conference (L4DC)*. PMLR, 2022, p. 13.

[26] S. N. Afriat, "Theory of maxima and the method of lagrange," *SIAM Journal on Applied Mathematics*, vol. 20, no. 3, pp. 343–357, 1971.

[27] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.

[28] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[29] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[30] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, 1950*. Berkeley and Los Angeles: University of California Press, 1951, pp. 481–492.

[31] C. Büskens and H. Maurer, "Sensitivity analysis and real-time control of nonlinear optimal control systems via nonlinear programming methods," in *Variational Calculus, Optimal Control and Applications: International Conference in Honour of L. Bittner and R. Klötzler*, Basel: Birkhäuser Basel, 1998.

[32] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE J. Robotics Autom.*, vol. 3, pp. 43–53, 1987.

[33] A. Allshire, M. Mittal, V. Lodaya, V. Makoviychuk, D. Makoviichuk, F. Widmaier, M. Wüthrich, S. Bauer, A. Handa, and A. Garg, "Transferring Dexterous Manipulation from GPU Simulation to a Remote Real-World TriFinger," *arXiv:2108.09779 [cs]*, 2021.

[34] N. Funk, C. Schaff, R. Madan, T. Yoneda, J. U. De Jesus, J. Watson, E. K. Gordon, F. Widmaier, S. Bauer, S. S. Srinivasa, T. Bhattacharjee, M. R. Walter, and J. Peters, "Benchmarking Structured Policies and Policy Optimization for Real-World Dexterous Object Manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 478–485, 2022.

[35] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The YCB object and Model set: Towards common benchmarks for manipulation research," in *2015 International Conference on Advanced Robotics (ICAR)*. Istanbul, Turkey: IEEE, 2015, pp. 510–517.

[36] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2019.