Learning Multi-Body Pushing Manipulation with Bayesian Optimization for MPPI Control

Yuzhou Chen
ME, ECE
University of Michigan
Ann Arbor, United States
yzc@umich.edu

Chenen Jin
Mechanical Engineering
University of Michigan
Ann Arbor, United States
jchenen@umich.edu

Zhiyin Xu
Mechanical Engineering
University of Michigan
Ann Arbor, United States
zhiyin@umich.edu

***Abstract*—In this project, we investigate two complementary approaches for improving robotic pushing manipulation. First, we learn the multi-body dynamics involved in indirect pushing tasks, where a robot manipulates a target object via an intermediate object, inspired by real-world scenarios with limited direct access. Using a learned dynamics model, we employ Model Predictive Path Integral (MPPI) control to plan effective pushing trajectories that guide the target object to its goal. Second, we focus on optimizing the hyperparameters of MPPI using Bayesian Optimization (BO) with Gaussian Processes (GPs). Our BO implementation, based on the GPyTorch library, models the performance landscape of MPPI across different hyperparameter configurations and guides exploration using acquisition functions like Expected Improvement (EI). We evaluate our method in a planar pushing environment with obstacles and compare it against three baselines: manually-tuned parameters, Covariance Matrix Adaptation Evolution Strategy (CMA-ES), and a scikit-learn based BO implementation. Our results show that learning multi-body dynamics significantly improves control performance, and that BO leads to better parameter tuning than other alternatives. This project demonstrates how GPs can be effectively leveraged in both dynamics learning and controller optimization for robotic manipulation.**

***Index Terms*—Bayesian Optimization, Gaussian Process, GPyTorch, Multi-Body, Dynamics,Pushing Manipulation**

I. Introduction

In modern automated warehouses and manufacturing systems, robotic manipulators are widely deployed to perform pick-and-place operations with high speed and reliability. However, in densely packed environments such as narrow shelving units or cluttered storage bins, direct access to target objects using a robotic gripper becomes challenging due to physical obstructions, limited maneuvering space, and occlusions. These constraints hinder the effectiveness of traditional grasping strategies that rely on clear access and precise end-effector positioning.

To overcome such limitations, indirect manipulation via intermediary objects—a process where a robot pushes the target item using a secondary tool or object—has emerged as a practical solution. Instead of grasping, the robot leverages a tool to push the target item toward a reachable or more favorable position, enabling subsequent grasping or further manipulation. This approach not only avoids unnecessary joint movements in constrained workspaces but also simplifies planning in cluttered scenes.

In this paper, we focus on the problem of pushing a target object to a designated goal region using an intermediate object controlled by the robot, with an emphasis on collision avoidance, alignment optimization, and task success guarantees. We formulate a cost-based model predictive control framework that incorporates goal proximity, obstacle avoidance, and push alignment into a unified objective function. We further demonstrate the effectiveness of our approach in simulated warehouse environments with narrow shelving and obstructed target items.

II. Related Work

Bayesian Optimization (BO) has been extensively applied to black-box optimization problems, where evaluations are expensive and gradients are unavailable. Frazier [1] provides a comprehensive tutorial on BO and its use of Gaussian Processes (GPs) as surrogate models for uncertainty estimation and decision making. BO has demonstrated strong sample efficiency compared to traditional random or grid search methods.

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is another widely used approach for black-box optimization in continuous domains. CMA-ES adapts the mean and covariance of a multivariate Gaussian distribution to explore the search space [2]. It is robust to noisy evaluations and local minima but typically requires more samples per update compared to BO. In this project, CMA-ES serves as a strong baseline for comparison against our GP-based Bayesian Optimization methods.

Our work builds upon these methods by integrating BO and CMA-ES into a real-time pushing control framework based on Model Predictive Path Integral (MPPI) control, and learning multi-body residual dynamics from scratch.

III. Method

A. Multi-Body Dynamics Learning

Learning an accurate dynamics of the multi-body interaction is the first and key step of this project, as the accuracy of the dynamics will directly affect the performance of the MPPI controller. To learn the system dynamics, a residual neural network combined with a multi-step loss were trained to obtain reliable predictions.

The main setup was similar to the methods used in [3].

Different from the task of pushing a single object, the multi-body task has the possibility of getting into undesired cases more frequently, such as the detachment of the intermediate object and target object and misalignment of these two objects. To avoid these extreme case, we discards the trajectories where the two objects are detached and added an alignment term to the cost function. By doing so, the neural network only learned the dynamics from the meaningful trajectories and the influence from the outliers was greatly reduced.

B. Gaussian Process

Gaussian Process (GP) regression is a non-parametric Bayesian approach for modeling unknown functions. Instead of producing a single deterministic output, a GP defines a distribution over possible functions, predicting both a mean and an uncertainty (variance) for every test input. Given training data, a GP uses a kernel function to model the correlation between points, allowing it to generalize effectively even with limited data. The kernel captures prior assumptions about the smoothness or periodicity of the underlying function.

To implement GP regression, we first compute the kernel matrix K between all pairs of training points, the kernel vector k_* between training points and the new test point, and the kernel scalar k_{**} for the test point itself. Using these quantities, the predictive mean and variance for the new point can be computed in closed form. Numerical stability is ensured by adding a small noise term to the diagonal of K. The overall procedure is summarized in Algorithm 1.

Algorithm 1 Gaussian Process Regression

Require: Training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, test input x^*, kernel function $k(\cdot, \cdot)$

Ensure: Predictive mean $\mu(x^*)$ and variance $\sigma^2(x^*)$

1: Compute kernel matrix K where $K_{ij} = k(x_i, x_j)$
2: Compute kernel vector $k_* = [k(x_1, x^*), \ldots, k(x_n, x^*)]^T$
3: Compute kernel scalar $k_{**} = k(x^*, x^*)$
4: Compute predictive mean: $\mu(x^*) = k_*^T (K + \sigma_n^2 I)^{-1} y$
5: Compute predictive variance: $\sigma^2(x^*) = k_{**} - k_*^T (K + \sigma_n^2 I)^{-1} k_*$

C. Bayesian Optimization

Bayesian Optimization provides an elegant and sample-efficient solution for black-box optimization problems by utilizing information from previous evaluations to determine the most promising point for the next query. This method is particularly well-suited for problems where evaluations are expensive, as is the case when tuning control parameters through physical or simulated robot interactions.

The core idea of Bayesian Optimization is to construct a probabilistic surrogate model of the unknown objective function. [1] This surrogate is typically a Gaussian Process (GP), which estimates both the mean and uncertainty of the objective value at any point in the parameter space. As new observations are added, the GP posterior becomes more accurate, allowing the optimizer to better distinguish between promising and uninformative regions.

The optimization process is guided by an acquisition function $a(x)$, which scores candidate points based on their expected utility. Common acquisition functions include Expected Improvement (EI) and Upper Confidence Bound (UCB). These functions balance the trade-off between exploring uncertain regions and exploiting known good areas.

The Bayesian Optimization algorithm can be summarized as pseudocode in Algorithm 2:

Algorithm 2 Bayesian Optimization Algorithm

Require: Objective function f, parameter space X, acquisition function S, surrogate model M

1: Initialize dataset $\mathcal{D} \leftarrow \{(x_i, f(x_i))\}_{i=1}^n$, where $x_i \in X$
2: **while** evaluation budget not exhausted **do**
3: Fit surrogate model M to $\mathcal{D}$ to obtain posterior $p(y \mid x, \mathcal{D})$
4: Select next query point: $x_{\text{next}} \leftarrow \arg\max_{x \in X} S(x; p(y \mid x, \mathcal{D}))$
5: Evaluate objective: $y_{\text{next}} \leftarrow f(x_{\text{next}})$
6: Update dataset: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x_{\text{next}}, y_{\text{next}})\}$
7: **end while**

D. Model Predictive Path Integral Control (MPPI)

Model Predictive Path Integral (MPPI) control is a sampling-based control strategy designed for high-dimensional, nonlinear, and potentially stochastic systems. It optimizes a sequence of control actions by drawing samples from a noisy distribution and evaluating them using a cost functional that captures task objectives. Compared to classical MPC formulations, MPPI provides a natural mechanism for integrating learned or black-box dynamics models, including those derived from neural networks.

The MPPI algorithm is outlined as Algorithm 3. At every control step, MPPI simulates K candidate control sequences over a finite horizon H, perturbed by Gaussian noise sampled from a covariance matrix Σ. Each sequence is evaluated based on cumulative running and terminal costs. The action update is then computed as a weighted average of the perturbations, where the weights are derived from the exponentiated negative trajectory costs scaled by a temperature parameter λ.

The performance of MPPI is highly dependent on its hyperparameters. The parameter λ influences the sensitivity to cost differences among trajectories, while Σ controls the magnitude of exploration. A poorly tuned λ may lead to either overly aggressive or overly conservative behavior. Therefore, selecting appropriate values for these parameters is essential for stable and effective control.

To address this, we apply Bayesian Optimization (Algorithm 2) to tune λ and Σ by minimizing a black-box cost function $J(\theta)$ that reflects task performance, such as final error, steps, energy expenditure, or success rate. The GP surrogate models the relationship between parameter configurations and

observed outcomes, allowing the optimizer to efficiently search the hyperparameter space.

Algorithm 3 Model Predictive Path Integral Control

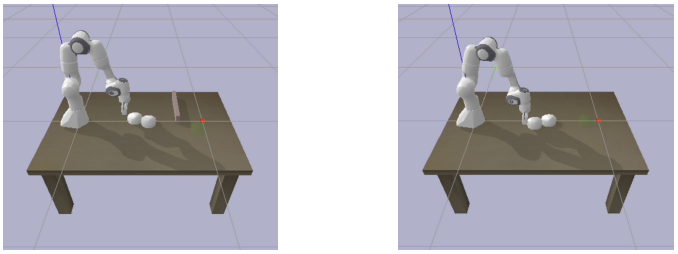
Require: Dynamics model F, sample count K, horizon T, initial control sequence $\{u_0, \ldots, u_{T-1}\}$, noise covariance Σ, temperature λ

while task not completed **do**
  $x_0 \leftarrow$ current state estimate
  for $k = 1$ to K **do**
    Sample perturbations $\{\epsilon_0^k, \ldots, \epsilon_{T-1}^k\} \sim \mathcal{N}(0, \Sigma)$
    Simulate trajectory using $x_{t+1}^k = F(x_t^k, u_t + \epsilon_t^k)$
    Compute cost $S^k = \sum_t q(x_t^k, u_t + \epsilon_t^k) + \Phi(x_T^k)$
  end for
  Normalize weights: $w^k = \frac{\exp(-S^k/\lambda)}{\sum_j \exp(-S^j/\lambda)}$
  for $t = 0$ to $T - 1$ **do**
    Update $u_t \leftarrow u_t + \sum_k w^k \epsilon_t^k$
  end for
  Apply u_0 to system and shift control horizon
end while

By embedding this control strategy within a Bayesian Optimization framework, we are able to automatically adjust MPPI hyperparameters to maximize performance on specific tasks. This process significantly reduces manual effort and improves adaptability to diverse or changing environments.

IV. Experimental Setup

In the experimental, we utilized the Panda Pushing environment as our simulation platform, where a robotic arm is tasked with pushing a disk-shaped object toward a specified target position, both in free space and in the presence of obstacles, as illustrated in Figure 1. For the controller, we employ a model predictive controller built upon a pretrained residual dynamics model, enabling accurate prediction of object motions over multiple steps.



(a) Free pushing (b) Pushing with obstacle

Fig. 1: Multi-body push under free and obstacle environment

A. Multi-Body Dynamics Learning

To accurately capture the dynamics of planar pushing tasks, we learn a multi-body dynamics model following a residual learning framework inspired by SAIN [3]. Our model builds upon a simple physics prior, predicting the residual state difference Δs rather than absolute states, which improves generalization across varied initial conditions.

a) Data Collection.: We collect random trajectories in simulation using the PandaPushingEnv environment. Each trajectory consists of states $\{s_t\}$ and actions $\{a_t\}$ collected by executing uniformly random actions. Each trajectory contains T transitions, organized as (s_t, a_t, s_{t+1}).

b) Training Setup.: We design a neural network-based residual model f_θ that predicts the state difference given current state and action:

$$\Delta \hat{s}_t = f_\theta(s_t, a_t), \tag{1}$$

and reconstruct the next state via

$$\hat{s}_{t+1} = s_t + \Delta \hat{s}_t. \tag{2}$$

The network consists of three fully connected layers with 100 hidden units each, activated by ReLU functions. We train the model using a single-step prediction loss based on SE(2) pose difference:

$$\begin{aligned}\mathcal{L}(s_{t+1}, \hat{s}_{t+1}) = &\,\text{MSE}(x, \hat{x}) + \text{MSE}(y, \hat{y}) \\ &+ r_g \left(\text{MSE}(\cos\theta, \cos\hat{\theta}) + \text{MSE}(\sin\theta, \sin\hat{\theta})\right)\end{aligned} \tag{3}$$

where r_g is the radius of gyration, computed from object size.

c) Multi-Step Prediction.: For long-horizon tasks, we extend the training to multi-step prediction. Given an initial state s_t, we predict over n steps by recursively applying the model, minimizing the discounted cumulative loss:

$$\mathcal{L}_{\text{multi}} = \sum_{k=0}^{n-1} \gamma^k \mathcal{L}(s_{t+k+1}, \hat{s}_{t+k+1}), \tag{4}$$

where γ is the discount factor set to 0.99.

d) Implementation Details.: We train the model using Adam optimizer with an initial learning rate of 0.001, batch size of 500, and employ an 80-20 train-validation split. Both single-step and multi-step datasets are processed into custom PyTorch Dataset classes for efficient loading.

B. Gaussian Process

To model the unknown objective function during optimization, we employ a Gaussian Process (GP) as a non-parametric surrogate model. A GP is fully specified by its mean function $m(x)$ and covariance function $k(x, x')$, and is defined as:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{5}$$

where $f(x)$ represents the modeled function at input x. In this work, we use the Radial Basis Function (RBF) kernel to define the covariance between two inputs x_i and x_j, given by:

$$k(x_i, x_j) = \exp\left(-\frac{d(x_i, x_j)^2}{2l^2}\right) \tag{6}$$

where $d(x_i, x_j)$ denotes the Euclidean distance, and l is the length-scale hyperparameter that controls the smoothness of the function.

Given a set of training inputs $X = \{x_1, \ldots, x_n\}$ and corresponding noisy observations $y = f(X) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$, the posterior predictive distribution for a new point x_* is Gaussian, with mean and variance given by:

$$\mu(x_*) = k(x_*, X)[K(X, X) + \sigma_n^2 I]^{-1} y \tag{7}$$

$$\sigma^2(x_*) = k(x_*, x_*) - k(x_*, X)[K(X, X) + \sigma_n^2 I]^{-1} k(X, x_*) \tag{8}$$

where $K(X, X)$ is the kernel matrix evaluated over all training points.

We train the GP model by maximizing the log marginal likelihood:

$$\log p(y|X) = -\frac{1}{2} y^\top [K(X,X) + \sigma_n^2 I]^{-1} y - \frac{1}{2} \log |K(X,X) + \sigma_n^2 I| - \frac{n}{2} \log 2\pi \quad (9)$$

using gradient-based optimization with a learning rate of 0.1 for 50 iterations.

During Bayesian Optimization, at each iteration, the GP is updated with new observations, and an acquisition function is evaluated to determine the next sampling point.

C. Bayesian Optimization

To systematically optimize the hyperparameters λ and Σ of the MPPI controller, we adopt Bayesian Optimization (BO), a sample-efficient approach well-suited for black-box, expensive-to-evaluate functions. Our implementation evaluates two common acquisition strategies: Expected Improvement (EI) and Upper Confidence Bound (UCB), both of which are designed to balance exploration and exploitation in the parameter space.

a) Expected Improvement (EI): The EI acquisition function estimates the expected gain in performance over the current best observation f_{best}. Given a candidate point x, the EI is defined as:

$$\begin{aligned} \text{EI}(x) &= \mathbb{E}\left[\max(f_{\text{best}} - f(x), 0)\right] \\ &= (f_{\text{best}} - \mu(x))\Phi(z) + \sigma(x)\phi(z) \end{aligned} \quad (10)$$

where $\mu(x)$ and $\sigma(x)$ are the posterior mean and standard deviation estimated by the GP, $z = \frac{f_{\text{best}} - \mu(x)}{\sigma(x)}$, and Φ, ϕ denote the cumulative distribution function and probability density function of the standard normal distribution, respectively. EI prioritizes regions with high expected utility based on both predicted value and uncertainty.

b) Upper Confidence Bound (UCB): UCB provides a principled way to navigate the exploration-exploitation trade-off by maximizing an upper confidence estimate of the objective function:

$$\text{UCB}(x) = \mu(x) - \beta \cdot \sigma(x) \quad (11)$$

Here, β is a tunable parameter controlling the emphasis on exploration. A higher β encourages sampling in areas of high uncertainty, while a lower β favors exploitation near known optima.

To benchmark the effectiveness of BO, we compare against the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), a state-of-the-art derivative-free optimization method for continuous domains. CMA-ES maintains a multivariate Gaussian distribution over candidate solutions, adapting its parameters based on the sampled fitness values.

We evaluate the optimization strategies on two planar pushing scenarios: one in an open workspace (free pushing), and the other with static obstacles (obstacle pushing). In each trial, two disk objects are initialized at randomized positions, and the robot must push one object to the other to reach a target configuration.

We compare the following four hyperparameter configurations:

- Manually-tuned parameters
- Parameters optimized using EI-based BO for 200 epochs
- Parameters optimized using UCB-based BO for 200 epochs
- Parameters optimized via CMA-ES

Each configuration is evaluated over multiple episodes, and the resulting performance is quantified using a custom cost function. To evaluate each run, we define the following cost function:

$$\text{Cost} = d_{\text{goal}} + \alpha N + \gamma(1 - \mathbb{I}_{\text{goal}}) \quad (12)$$

where:

- d_{goal}: Euclidean distance between the object and the target at the final timestep
- N: Number of control steps taken
- $\mathbb{I}_{\text{goal}}$: Indicator function equal to 1 if the goal is reached, 0 otherwise
- α: Weighting coefficient for trajectory length penalty
- γ: Penalty for failing to reach the goal

This cost formulation penalizes solutions with large final errors, excessive control effort, and failure to complete the task. Minimizing this cost leads to shorter, more accurate, and successful control trajectories.

V. Result and Discussion

A. Multi-body Dynamics

We first evaluate the performance of the learned residual dynamics model used to predict the object's planar motion under pushing actions. A total of 500 random trajectories were collected in the Panda Pushing environment, with each trajectory consisting of multiple state-action transitions. The dataset was split into training and validation sets with an 80-20 ratio. We trained the residual dynamics model for 10,000 epochs, reducing the training loss to 0.0002 and the evaluation loss to 0.0004, indicating accurate modeling of the object's planar behavior.

Qualitatively, the model accurately captures small displacements and rotations of the pushed object, even when generalizing to unseen initial conditions. The residual formulation, predicting the change in state rather than the absolute next state, was particularly beneficial for stabilizing long-horizon rollouts. This aligns with prior observations in [3], where learning residuals over physical priors improves generalization and robustness.

Quantitatively, multi-step prediction experiments show that the accumulated prediction error grows sublinearly with horizon length, validating the model's ability to perform multi-step forecasting. The learned dynamics model enables the MPPI controller to generate effective pushing trajectories both in free space and in the presence of obstacles, demonstrating that model-based planning is feasible even with approximate learned dynamics.

B. Bayesian Optimization with Gaussian Process

During training, we configured the number of epochs as 200, which is defined as the number of times the planar pushing task. Each configuration yielded a distinct set of optimized hyperparameters. To assess the quality of these hyperparameters, we performed 50 independent test trials for each set. For each trial, we record the success rate of the task, the number of steps required to complete the task, and the corresponding cost. The evaluation results for the hyperparameters obtained after 200 epochs of training are summarized in Table I. In this context, "Free Pushing" refers to a push task without obstacles and with a fixed target position, whereas "Obstacle Pushing" denotes a more challenging setting involving fixed obstacles and a fix target location.

We evaluate the performance of different hyperparameter optimization methods across two planar pushing scenarios: a free pushing environment without obstacles, and a more challenging obstacle pushing environment with fixed obstacles and target locations. Each optimization strategy—Bayesian Optimization with Expected Improvement (BO-EI), Bayesian Optimization with Upper Confidence Bound (BO-UCB), CMA-ES, and manual tuning—was assessed over 50 test trials per setting. The metrics considered include success rate, average number of steps to complete the task, and trajectory cost.

1) Free Pushing Environment: The results for the free pushing scenario are summarized in Fig. 2. Bayesian Optimization with UCB achieved the highest success rate (96%), followed closely by B0-EI (88%). Both methods significantly outperformed CMA-ES (80%) and manual tuning (36%). In terms of average trajectory cost and number of steps, the Bayesian optimization methods consistently resulted in lower values, indicating more efficient and direct pushes toward the target.

Notably, the failure rates for manual tuning were substantial, with frequent deviations from the desired trajectories. This highlights the difficulty of manually selecting MPPI hyperparameters, even in relatively simple environments. CMA-ES improved over manual parameters but exhibited larger variability in performance compared to Bayesian optimization.

2) Obstacle Pushing Environment: The performance results in the obstacle pushing setting are shown in Fig. 3. As expected, the success rates decreased for all methods due to the added complexity of navigating around obstacles. Nonetheless, B0-EI and B0-UCB maintained significantly higher success rates (78% for both) compared to CMA-ES (38%) and manual tuning (18%).

Furthermore, both Bayesian optimization methods achieved lower average costs and fewer steps, indicating their ability to find more efficient control sequences even under challenging conditions. In contrast, CMA-ES and manual tuning often resulted in failed trajectories, as indicated by a large number of failed trials marked in the plots. These failures suggest difficulties in adapting to the constrained environment when hyperparameters were not optimally tuned.

3) Discussion: Across both environments, Bayesian Optimization demonstrates a clear advantage over manual tuning and CMA-ES in terms of success rate, efficiency (steps taken), and cost minimization. Between the two acquisition strategies, BO-UCB slightly outperforms BO-EI in free pushing, while they perform comparably in the obstacle pushing scenario. These results suggest that carefully modeling the hyperparameter landscape using Gaussian Processes enables more effective exploration and exploitation compared to black-box evolutionary strategies. Moreover, the robustness of Bayesian Optimization under varying task difficulties highlights its potential for tuning MPPI controllers in real-world applications where environmental conditions are dynamic and unpredictable.

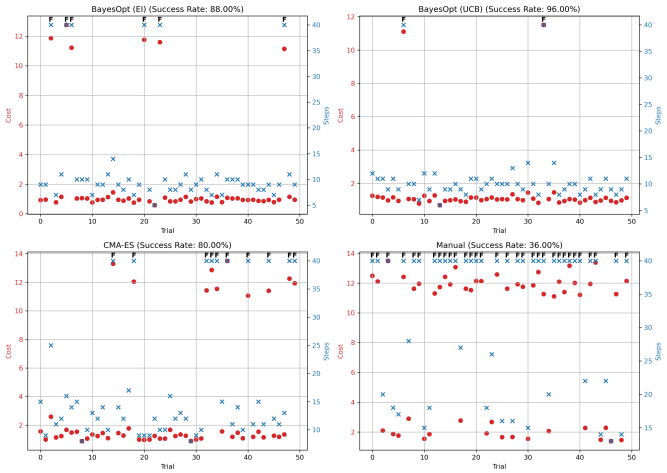


Fig. 2: Result of pushing task with free-space

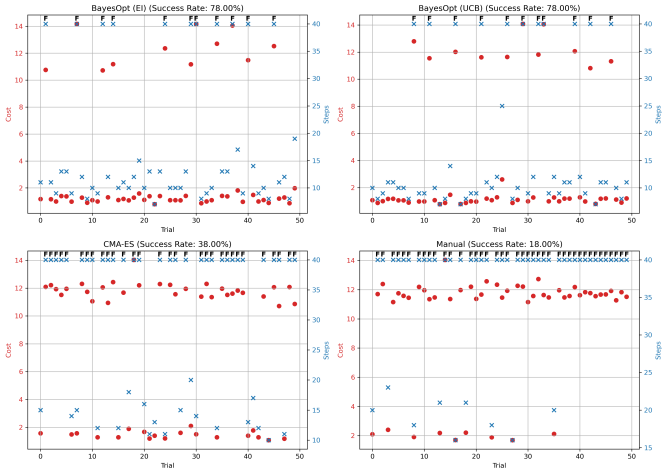


Fig. 3: Result of pushing task with obstacle

Table I presents the optimization results across 200 epochs for free and obstacle pushing tasks. BO-EI and BO-UCB, our proposed methods based on Bayesian Optimization, achieved lower average costs and higher success rates compared to baseline methods (CMA-ES and Hand Tuned) in both settings. Although CMA-ES achieved slightly better cost in free pushing, BO-based methods demonstrated a strong balance between cost, step efficiency, and success rate, confirming their effectiveness and robustness for complex manipulation tasks.

TABLE I: Optimization results of 200 epochs

Algorithm	Free Pushing			Obstacle Pushing		
	Cost	Step	Goal (%)	Cost	Step	Goal (%)
BO-EI	0.90	9.20	88	**3.02**	**10.50**	**78**
BO-UCB	0.77	**9.05**	**96**	3.72	10.90	**78**
CMA-ES	**0.68**	11.90	80	4.76	14.80	38
Hand Tuned	0.78	18.10	36	4.12	20.00	18

Table II presents the runtime comparison between different optimization methods for free and obstacle pushing tasks. BO-EI and BO-UCB are our proposed methods based on Bayesian Optimization with Expected Improvement and Upper Confidence Bound, respectively, while CMA-ES and Hand Tuned serve as baselines. Although CMA-ES and Hand Tuned show slightly faster runtimes, the differences across all methods are small, confirming that our approaches are efficient and suitable for real-time robotic applications.

TABLE II: Runtime comparison (in seconds) for free and obstacle pushing

Algorithm	**Free Pushing**	**Obstacle Pushing**
BO-EI	0.3891	0.4122
BO-UCB	0.4602	0.4255
CMA-ES	0.3311	0.3608
Hand Tuned	0.3371	0.3706

VI. Conclusion

In this project, we investigated a unified framework combining learned multi-body dynamics and Bayesian Optimization for model-based pushing manipulation. We trained a residual dynamics model using 500 collected trajectories, achieving low prediction error after 10,000 epochs, and integrated it with an MPPI controller to perform multi-step pushing tasks.

To optimize the MPPI hyperparameters, we applied Bayesian Optimization with Gaussian Processes, employing Expected Improvement (EI) and Upper Confidence Bound (UCB) acquisition strategies. Compared to manual tuning and CMA-ES baselines, our BO-based methods achieved higher success rates, lower costs, and more efficient pushing trajectories in both free-space and obstacle-cluttered environments.

The results demonstrate that combining learned dynamics with sample-efficient hyperparameter optimization can significantly improve the robustness and efficiency of contact-rich manipulation tasks, paving the way for future extensions toward real-world robotic deployment.

References

[1] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[2] N. Hansen, "The cma evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[3] A. Ajay, M. Bauza, J. Wu, N. Fazeli, J. B. Tenenbaum, A. Rodriguez, and L. P. Kaelbling, "Combining physical simulators and object-based networks for control," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3217–3223.

Appendix: Code Repository

The source code for this project is available at: https://github.com/yzyzchen/Learning-Multi-Body-Manipulation-with-Bayesian-Optimizationl.git