Sampling-Based Model Predictive Control Leveraging Parallelizable Physics Simulations

Corrado Pezzato, Chadi Salmi, Elia Trevisan, *Graduate Student Member, IEEE*, Max Spahn, *Member, IEEE*, Javier Alonso-Mora, *Senior Member, IEEE*, and Carlos Hernández Corbato

***Abstract*—We present a sampling-based model predictive control method that uses a generic physics simulator as the dynamical model. In particular, we propose a Model Predictive Path Integral controller (MPPI) that employs the GPU-parallelizable IsaacGym simulator to compute the forward dynamics of the robot and environment. Since the simulator implicitly defines the dynamic model, our method is readily extendable to different objects and robots, allowing one to solve complex navigation and contact-rich tasks. We demonstrate the effectiveness of this method in several simulated and real-world settings, including mobile navigation with collision avoidance, non-prehensile manipulation, and whole-body control for high-dimensional configuration spaces. This is a powerful and accessible open-source tool to solve many contact-rich motion planning tasks.**

***Index Terms*—Optimization and Optimal Control, Contact Modeling, Whole-Body Motion Planning and Control, Model Predictive Path Integral Control.**

I. Introduction

As ROBOTS become increasingly integrated into our daily lives, their ability to navigate and interact with the environment is becoming more important than ever. From collision avoidance to moving obstacles out of the way to pick up some objects, robots must be able to plan their motions while accounting for contact with their surroundings. At the same time, robotic platforms require many Degrees Of Freedom (DOF) to achieve agile and dexterous movements. All this poses many challenging problems to motion planners, such as collision-free navigation in complex and dynamic environments, high DOF mobile manipulation, contact-rich tasks such as picking and pushing, and in-hand manipulation. Solutions to these challenges exist but are often specialized and not easily transferable to different scenarios. Learning-based approaches, for example, can leverage physics simulators to train policies for complex tasks but require extensive training and resources. For instance, [1] took years to develop, utilizing 6144 CPU cores and 50 hours of training to learn a policy for in-hand cube manipulation.

Received 13 November 2024; accepted 6 January 2025. Date of publication 28 January 2025; date of current version 10 February 2025. This article was recommended for publication by Associate Editor E. M. Hoffman and Editor L. Pallottino upon evaluation of the reviewers' comments. This work was supported in part by Ahold Delhaize and and in part by the project of the Netherlands Organization for Scientific Research (NWO), domain Science (ENW) through the project "Sustainable Transportation and Logistics over Water: Electrification, Automation, and Optimization (TRiLOGy). " *(Corrado Pezzato, Chadi Salmi, and Elia Trevisan contributed equally to this work.) (Corresponding author: Elia Trevisan.)*

The authors are with the Cognitive Robotics Department, TU Delft, 2628 CD Delft, The Netherlands (e-mail: e.trevisan@tudelft.nl; j.alonsomora@tudelft.nl; c.h.corbato@tudelft.nl).

Code and videos: https://autonomousrobots.nl/paper_websites/isaac-mppi.

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2025.3535185, provided by the authors.

Digital Object Identifier 10.1109/LRA.2025.3535185

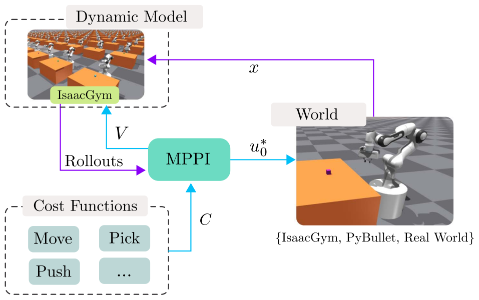


Fig. 1. Scheme of the proposed method using IsaacGym as the dynamic model for MPPI. At each time step, IsaacGym is reset to the current world's state x, and random input sequences V are applied for the horizon T, to every environment. MPPI uses the resulting rolled-out trajectories to approximate the optimal control u_0^* given a cost function C.

On the other hand, model-based approaches like Model Predictive Control (MPC) can solve challenging tasks [2]. However, MPC often relies on constrained optimization, requiring constraint simplifications, precise modeling, and ad-hoc solutions to handle discontinuous dynamics in contact-rich tasks [3], [4]. While utilizing motion memory for warm-starting optimization can enhance performance [5], [6], the above limitations still persist. Recently, Model Predictive Path Integral (MPPI) control [7] and its information-theoretic counterpart [8] addressed optimal control problems via importance sampling, mitigating challenges tied to constrained optimization algorithms dealing with non-convex constraints and discontinuous dynamics. However, substantial modeling remains necessary.

In this paper, we propose a training-free model-based framework for real-time control of complex systems, where one designs only a cost function, not the problem's dynamics and contact models. We introduce the idea of using a general GPU-parallelizable physics simulator, IsaacGym [9], as the dynamic model for MPPI. This creates a robust framework that generalizes to various tasks. An overview is given in Fig. 1.

A. Related Work

This section provides an overview of selected works focusing on motion planning and contact-rich tasks in robotics. Motion planning pipelines are categorized as global and local motion planning [10]. Local motion planning encompasses approaches like operational space control, geometric methods such as Riemannian Motion Policies [11] and Optimization Fabrics [12], [13], and receding-horizon optimization formulations like Model Predictive Control (MPC) [14] that may incorporate learned components [15]. Most MPC algorithms rely on constrained optimization and assume smooth dynamics. However, contact-rich tasks pose challenges due to their non-smooth and hybrid nature, involving sticking and sliding frictions or entering contacts, requiring extensive modeling and ad-hoc solutions for pushing tasks [3], [4].

In contrast, Model Predictive Path Integral (MPPI) control [7], [8] is a sampling-based MPC approach that approximates optimal control via parallel sampling of input sequences. MPPI is gradient-free and well-suited for systems with non-linear, non-convex, discontinuous dynamics and cost functions. It has successfully controlled high-degree-of-freedom manipulators in real-time [16], incorporating self-collision avoidance using trained neural networks and collision-checking functions [17]. However, these approaches have limited interaction with the environment. In [18], the authors propose ensemble MPPI, a variation that handles complex tasks and adapts to parameter uncertainty. Still, the task modeling remains unclear, and no open-source implementation is available.

To alleviate the problem of explicit modeling, some works have addressed the use of physics simulators for sampling-based MPC. In [19], the authors use the RaiSim simulator to sample waypoints for foot placement of a quadruped. Moreover, Howell et al. [20] proposed a sampling-based MPC method that employs MuJoCo [21] as a dynamic model for rolling out sampled input sequences. This offloads modeling efforts to the physics engine, simplifying controller design. However, MuJoCo's parallelization capabilities are constrained by the number of CPU threads, limiting real-time performance when many samples are required to solve a task. Moreover, results are presented only in simulation.

A high number of samples is particularly crucial in tasks such as non-prehensile manipulation with robot manipulators. Traditional approaches often involve sampling end-effector trajectories on a plane, relying on additional controllers for robot actuation and learned models for predictions [22], [23]. For instance, Arruda et al. [22] use a forward-learned model trained on 326 real robot pushes. This model is employed by an MPPI controller to plan push manipulations as end-effector trajectories. Cong et al. [23] train a Long Short-Term Memory-based model to capture push dynamics using a dataset of 300 randomized objects. End-effector trajectories are sampled within a rectangular 2D workspace. Both methods require a separate controller to convert cartesian motions into joint commands, and both perform push manipulation through a sequence of pushes, resulting in discontinuous motion. These methods are not easily transferable to other robots, particularly non-holonomic mobile pushing.

B. Contributions

This paper presents a novel open-source implementation of Model Predictive Path Integral (MPPI) control with a generic physics simulator as the dynamical model. This enables the method to solve many contact-rich motion planning problems. The two key contributions of this work are:

- The integration of the MPPI controller with the GPU-parallelizable simulator IsaacGym, distinguishing our approach from prior works in MPPI. Our method facilitates collision checking and contact-rich manipulation tasks leveraging the contact models and rigid body interactions included in the simulator without requiring gradients. Our solution allows smooth real-time control of real-world systems with high degrees of freedom, efficiently computing hundreds of rollouts in parallel.
- A versatile method applicable to various motion planning challenges, including collision avoidance, prehensile and non-prehensile manipulation, and whole-body control with diverse robots. We provide an open-source implementation that can be readily reused and extended to heterogeneous robots and tasks.

We perform many contact-rich tasks with several robotic platforms and real-world experiments. We include omnidirectional and differential drive robots and fixed or mobile manipulators and compare against many specialized baselines.

II. Sampling-Based MPC Via Parallelizable Physics Simulations

In this section we describe the integration of MPPI with IsaacGym, which enables real-time control of complex contact-rich robotic systems with minimal modeling.

A. Background Theory on MPPI

In this section, we give an overview of the background theory of MPPI. For more theoretical insights, please refer to the original publications [7], [8]. MPPI is a method to solve stochastic optimal control problems for discrete-time dynamical systems such as

$$x_{t+1} = f(x_t, v_t), \quad v_t \sim \mathcal{N}(u_t, \Sigma), \tag{1}$$

where the nonlinear state-transition function f describes how the state x evolves over time t with a control input v_t. MPPI samples K noisy input sequences V_k. These sequences are then applied to the system to simulate K state trajectories Q_k, $k \in [1, K]$, over a time horizon T:

$$Q_k = [x_0, f(x_0, v_0), \ldots, f(x_{T-1}, v_{T-1})]. \tag{2}$$

Given the state trajectories Q_k and a designed cost function C to be minimized, the total state-cost S_k of an input sequence V_k is computed by functional composition $S_k = C(Q_k)$. Then, each rollout is weighted by importance sampling weights w_k, computed via an inverse exponential of S_k with tuning parameter β, normalized by η. The minimum sampled cost $\rho = \min_k S_k$ is subtracted for numerical stability, leading to:

$$w_k = \frac{1}{\eta} \exp\left(-\frac{1}{\beta}(S_k - \rho)\right), \quad \sum_{k=1}^{K} w_k = 1 \tag{3}$$

The parameter β is also known as *inverse temperature*. The weights are then used to compute the approximate optimal control input sequence U^*:

$$U^* = \sum_{k=1}^{K} w_k V_k \tag{4}$$

Equations (3) and (4) demonstrate the approach taken to approximate the optimal control. Equation (4) represents a weighted average of sampled control inputs, while (3) assigns exponentially higher weights to less costly inputs. The first input u_0^* of the sequence U^* is applied to the system. Then the process is repeated.

B. Proposed Algorithm

We now describe how we use MPPI with IsaacGym, summarized in Algorithm 1. We initialize an input sequence U_{init} as a vector of zeroes with a length of T, where T is the time horizon in steps. We then sample K sequences of additive input noise $\mathcal{E}_k$ for exploring the input space around U_{init}. The key concept is that, instead of explicitly defining a nonlinear transition function f, we use IsaacGym to compute the next state x_{t+1} given x_t and control input v_t. This is done by reading the current state of the environment, resetting the state of the simulator to the observed values, and then applying the noisy control input sequence to simulate the state trajectories in IsaacGym. Note that these K state trajectories can be computed independently of each other. We use this property to forward and simulate all the rollouts in parallel, leveraging the parallelization capabilities of IsaacGym. Instead of sampling from a Gaussian distribution, we follow the strategy of a recent paper [16] that proposes to sample *Halton Splines* instead for better exploration and smoother trajectories. Similar to [16], we fit B-Splines to inputs sampled from a Halton sequence using standard Python modules and then we evaluate the spline at regular intervals to retrieve $\mathcal{E}_k$. Unlike [16], we do not update the variance of the sampling distribution. Instead, we keep it as a tuning parameter, constant during execution. Updating the variance as in [16] can lead to better convergence to a goal, but it also leads to stagnation of the control over time, which is harmful in the contact-rich tasks considered in this paper. Once the task begins, we reset our K simulation environments on IsaacGym to the current observed *world* state x. In parallel, we can now roll out the sampled input sequences V_k into state trajectories Q_k using K simulation environments on IsaacGym and compute their corresponding cost S_k using the designed *cost function* C. The cost is discounted over the planning horizon T by a factor γ [16]:

$$S_k = \sum_{t=0}^{T-1} \gamma^t C(x_{t,k}, v_{t,k}) \tag{5}$$

Next, we can compute the *importance sampling* weights w_k as in (3). The normalization factor η is a useful metric to monitor, as it indicates the number of samples assigned significant weights. We use this to tune β for the next iteration such that η is maintained within an upper and lower bound:

$$\beta_{t+1} = \begin{cases} 0.9\beta_t & \text{if } \eta > \eta_{max} \\ 1.2\beta_t & \text{if } \eta < \eta_{min} \\ \beta_t & \text{otherwise} \end{cases} \tag{6}$$

Empirically, we observed in all performed tasks that setting $5 < \eta < 10$ is a good balance for smooth behavior. Finally, an approximation of the optimal control sequence U^* can now be computed via a weighted average of the sampled inputs (4). U_{init} is now updated with U^*, time-shifted backward of one timestep

Algorithm 1: Proposed Approach.

```
 1: Initialize:
      $U_{init} = [0, \dots, 0]$                          ▷ $U_{init} \in \mathbb{R}^T$
      $\mathcal{E}_k \leftarrow sampleHaltonSplines()$     ▷ $k = 1 \dots K$
 2: while $taskNotDone$ do
 3:   $x \leftarrow observeEnvironment()$
 4:   $resetSimulations(x)$
 5:   for $k = 1 \dots K$                                 ▷ in parallel do
 6:     $V_k = U_{init} + \mathcal{E}_k$
 7:     $[Q_k, S_k] \leftarrow computeRolloutCost(V_k, \gamma)$
 8:     $w_k \leftarrow importanceSampling$                ▷ (3)
 9:   $\beta \leftarrow updateBeta(\beta, \eta)$              ▷ (6)
10:   $U^* = \sum_{k=1}^{K} w_k V_k$
11:   $U_{init} \leftarrow timeShift(U^*)$
12:   $applyInput(u_0^*)$
```

so that it can be used as a warm-start for the next iteration, $U_{init} = [u_1^*, \dots, u_{T-1}^*, u_{T-1}^*] \in \mathbb{R}^T$. The second last input in the shifted sequence is propagated to the last input as well. From the sequence U^*, only the first input u_0^* is applied to the system, and the next iteration starts.

C. Exploiting the Physics Simulator Features

IsaacGym provides useful information and general models that are particularly useful for robot control in contact-rich tasks. Besides being useful to simulate the physical interaction of rigid bodies, we leverage IsaacGym for *collision checking* and *tackling model uncertainty* with domain randomization.

1) Collision Checking: Collision checking in robotics can be challenging for a number of reasons, one of them being computational complexity. This is particularly true if the task requires continuous collision checking as the robot moves in dense environments with complex object shapes. To overcome this problem, approximations are often introduced with the convexification of the space. However, this requires several heuristics and can hinder robot motions in complex scenes.

Instead, we propose to tackle the problem of collision checking by using the already available *contact forces tensor* from IsaacGym, which is available for each simulation step. To avoid collisions, we then define a cost function proportional to the contact forces for the MPPI:

$$C_{coll} = \omega_c \sum F_{obst}, \tag{7}$$

where F_{obst} are the contact forces exerted on the different obstacles. This allows us to perform continuous collision checking at each time step over the horizon T, with arbitrary complex shapes. By heavily penalizing contacts with obstacles, the robot will avoid collisions. On the other hand, by relaxing the weight ω_c one can allow for certain contacts required for the task, such as rolling a ball against a wall (Section III-C2b).

2) Tackling Model Uncertainty: IsaacGym is designed to easily support domain randomization. We use this feature to randomize the object properties in each environment in case of contact-rich tasks, such that uncertainty is incorporated in every rollout for the MPPI. Effectively, this allows to account for uncertainty in environment perception. Specifically, starting from

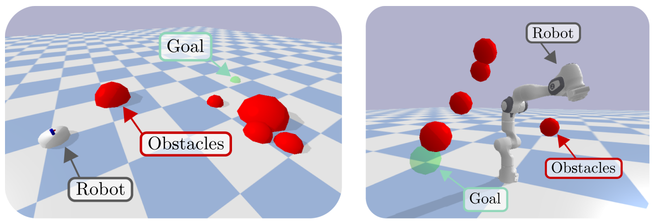


Fig. 2. Examples for pure motion planning benchmark setups. Left: point robot with 3 DOF. Right: manipulator with 7 DOF.

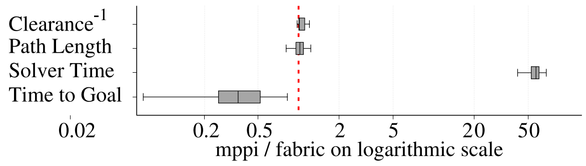
nominal physics properties, in every rollout objects are spawned with uncertainty on mass and friction nominal values, sampled from a uniform distribution. Additionally, the object size is also randomized with additive Gaussian noise, see Section III for experiment-specific details. Therefore, every simulation is different from the others, and all simulations are different from the *world* such that we can account for model mismatch. In a sense, we perform a sort of domain randomization in real-time to address the challenge of model uncertainty and imperfect perception.

III. Experiments

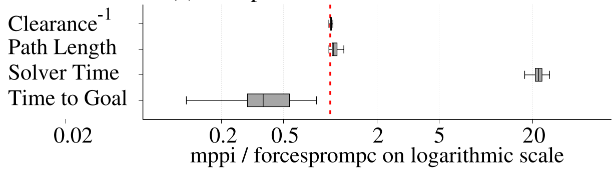
We perform several experiments in three different categories: 1) *motion planning and collision avoidance*, 2) *whole-body control* of high DOF systems in contact-rich settings, and 3) *non-prehensile manipulation*. Experiments and simulations are conducted on an Alienware Laptop with Nvidia 3070 Ti graphics card. The software implementation (https://autonomousrobots.nl/paper_websites/isaac-mppi) consists of our open source Python package that can easily be installed, tested, and extended to new robots and tasks. In real-world tests, we used a Robot Operating System (ROS) wrapper to connect the robot to the planner and a motion capture system to determine the pose of manipulated objects. Our implementation allows for position, velocity, and torque control. In this paper, all robots are velocity-controlled except for the mobile manipulator in Section III-B, which is torque-controlled.

A. Motion Planning and Collision Avoidance

We compare the performance of the proposed method in a pure local motion planning setting, i.e. no interaction with the environment. This aims to showcase the fact that our method is comparable to state-of-the-art techniques when no contact is involved. The main focus is the quantitative analysis of the method compared to two baselines, specifically optimization fabrics as presented in [12], [13] and a simple MPC formulation solved with ForcesPro [24]. We make use of an already available benchmark setup, the *localPlannerBench* [25]. We present results for two cases, namely a holonomic robot, and a robotic arm (Franka Emika Panda). For all experiments, we randomize five obstacles and the goal positions in $N = 100$ runs, see Fig. 2 for some examples. Solutions by the three methods are assessed using four metrics, e.g. time to reach the goal, path length, solver time, and minimum clearance. The compared methods show minimal differences in path length clearance for both examples (Fig. 3). However, our method consistently reaches the goal

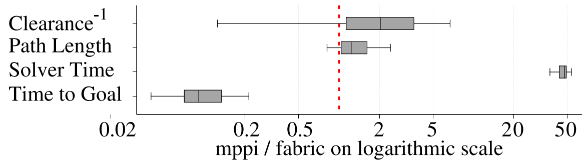


(a) Comparison with Fabrics

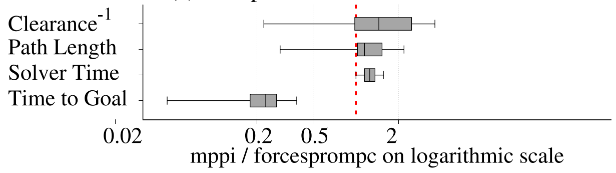


(b) Comparison with ForcesPro MPC

Fig. 3. Results in pure motion planning problems for point robot with 3 DOF.



(a) Comparison with Fabrics



(b) Comparison with ForcesPro MPC

Fig. 4. Results in pure motion planning problems for a robot manipulator with 7 DOF.

TABLE I
Summary of Motion Planning Experiments

Metric	Robot	Fabric	ForcesPro MPC	MPPI
Average Solver Time	Point	**1.0**ms	2.5ms	55ms
	Panda	**1.4**ms	51ms	63ms
Average Time to Goal	Point	7.4s	6.1s	**2.7**s
	Panda	9.6s	4.2s	**0.8**s

faster (Figs. 3 and 4, and Table I). This is attributed to the perfect representation of the robot's collision shapes used in our method, compared to the enclosing spheres in the ForcesPro MPC and optimization fabrics. It should be noted that our approach incurs higher computational times (Table I) due to the physics simulations performed by IsaacGym. Despite this, our method remains competitive in motion planning applications and offers significant advantages in contact-rich tasks, as demonstrated in the following sections.

B. Prehensile Manipulation With Whole-Body Control

Our approach scales well with the complexity of the robot. In Fig. 5, the task is to relocate an object from a table to an $[x, y, z]$ location using a mobile manipulator with 12 DOF.

Although this is arguably a complex task for a robot, which usually requires manual engineering of a sequence of movements, such as navigation to a specific base goal, and pre-post

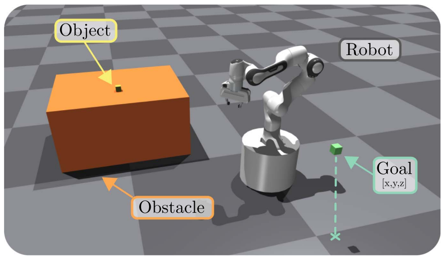


Fig. 5. Whole-body motion of a mobile manipulator moving a cube from initial to a desired location.

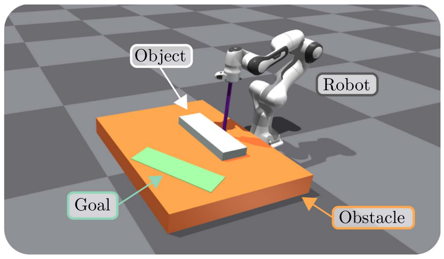


Fig. 6. Example of non-prehensile push task with a 7-DOF robot arm using an object from [23].

grasps, the solution is rather simple with our method. In fact, we specify the following cost function for the task:

$$C_{pick} = C_{dist} + C_{pose} + C_{coll} + C_{vel}, \tag{8}$$

where we consider the Euclidean distance of the end-effector to the object and the object to the goal: $C_{dist} = \omega_t ||p_{EE} - p_O|| + \omega_{O_p} ||p_G - p_O||$. We give an incentive to keep the robot in a comfortable pose by penalizing deviations from a desired arm and gripper pose, end-effector orientation, as well as imposing a minimum end-effector height $C_{pose} = C_{Parm} + C_{Pgrip} + C_{Oee} + C_{Hee}$. We minimize collisions penalizing the forces on the table C_{coll}. Lastly, we penalize high arm and base velocities $C_{vel} = C_{Varm} + C_{Vbase}$ since, in this experiment, we torque-control the robot. By sampling all the DOF at once, including the base and the gripper, we achieve a fluid motion from start to end with no added heuristics for pick positions. We performed ten pick-and-deliver tasks, and the time taken was 15.67 ± 7.21 s. The high standard deviation is because sometimes the cube falls, but the robot can recover by picking it up again from the floor.

For smooth whole-body motions of high DOF systems like this, many samples are required. Empirically, when the number of samples exceeds 50, a GPU pipeline is computationally cheaper than a CPU and scales better. Using IsaacGym, we can compute all the 750 samples required for mobile manipulation in parallel, computing the next control input online at 25 Hz.

C. Non-Prehensile Manipulation

One advantage of using a physics simulator is that one can leverage generic physics rules for contacts, thus eliminating the need for learning or engineering specialized contact models. We demonstrate this in non-prehensile manipulation tasks involving a 7-DOF arm (Fig. 6) and two different mobile robots (Figs. 7, 8, and 9). In Section III-C1, we apply our method to the two pushing tasks tackled in [22], [23], and we compare with their

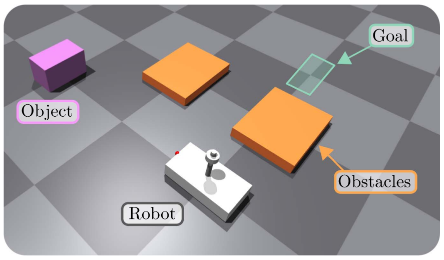


Fig. 7. Non-prehensile task using an omnidirectional base. $\omega_t = 0.2, \omega_{O_p} = 2, \omega_{O_r} = 3, \omega_a = 0.6, \omega_c = 10, T = 8, dt = 0.04, K = 300$.

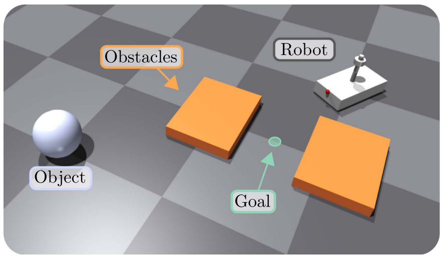


Fig. 8. Rolling ball non-prehensile pushing. Goal: Ball placement between two obstacles. $\omega_t = 0.2$, $\omega_{O_p} = 0.1$, $\omega_{O_r} = 0$, $\omega_a = 0.1$, $\omega_c = 0.001$, $T = 8$, $dt = 0.04$, $K = 300$.

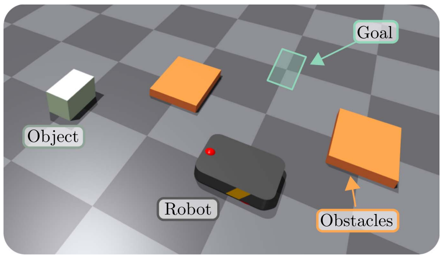


Fig. 9. Non-prehensile differential drive pushing. Same task as omnidirectional base. $\omega_t = 0.1$, $\omega_{O_p} = 2$, $\omega_{O_r} = 3$, $\omega_a = 0.6$, $\omega_c = 100$, $T = 12$, $dt = 0.04$, $K = 400$.

final results. Additionally, in Section III-C2, we demonstrate the ease of transferring our approach to different robots, including differential-drive.

1) Comparison With Baselines for Pushing With a Robot Arm: We consider two baselines for non-prehensile pushing. In the first one, [22] tackles the problem of pushing a relatively small object to a target pose with either 0 (Pose 1) or 90 deg (Pose 2). They also consider sequences of push actions starting far from the object. In the second baseline, [23] considers 5 relatively big objects and assumes the robot's end effector is close to the object during execution. Since we do not have access to the same hardware, and the authors of the considered baselines do not provide their models and data, we only compare against their final results. We set up our simulation to match as close as possible the tasks in the baseline using the available information from the papers. Finally, we tune our method for the two tasks separately for a fair comparison with the individual baselines. The approach in [22] utilizes an MPPI in combination with a learned model for predicting pushing effects on an object. The authors sample 2D end-effector trajectories and then rely on inverse kinematic solvers, achieving push manipulation as a

TABLE II
SUMMARY OF COMPARISON WITH [22]

Approach	Start pose	Final cost
Baseline [22]	Pose 1	0.057
	Pose 2	0.079
Ours (sim)	Pose 1	**0.029** ± 0.09
	Pose 2	**0.03** ± 0.12

TABLE III
SUMMARY OF COMPARISON WITH [23]

Approach	Metric	Object				
		A	B	C	D	E
Baseline [23]	Success [%]	93.5	90.9	93.9	91.6	**89.5**
Ours (sim)	Success [%]	**100**	**93.3**	**96.7**	**100**	66.7

sequence of disconnected pushes. In contrast, we use MPPI to sample the control input directly as joint velocities in IsaacGym. By doing so, we achieve smooth continuous pushes where end-effector repositioning emerges naturally, and learning is not required. The cost function to be minimized for the task is:

$$C_{push} = C_{dist} + C_{push\ align} + C_{ee\ align}. \tag{9}$$

C_{dist} has the weighted distance *robot-object*, and *object-goal:*

$$C_{dist} = \omega_t||p_R - p_O|| + \omega_{O_p}||p_G - p_O|| + \omega_{O_r}||\psi_O - \psi_G||,$$

where p_G and ψ_G are the goal's position and orientation, while p_R and p_O denote the end-effector tip and block positions, respectively. The cost function $C_{push\ align}$ promotes keeping the object between the robot and the goal. It is computed as $cos(\alpha) + 1$, where α is the angle between the *robot-object* $(p_R - p_O)$ and *goal-object* $(p_G - p_O)$ vectors and $+1$ is added to make the cost term always non-negative [26]:

$$C_{push\ align} = \omega_a \left(\underbrace{\frac{(p_R - p_O)\cdot(p_G - p_O)}{||p_R - p_O||||p_G - p_O||}}_{cos(\alpha)} + 1 \right). \tag{10}$$

We promote the end-effector to maintain a downward orientation at height d_h using pitch θ and roll ϕ:

$$C_{ee\ align} = \omega_{ee_r}||[\phi, \theta] - [0, 0]|| + \omega_{ee_h}||p_{R_z} - d_h||.$$

The cost is minimized when the end-effector is close to the block at a certain height and orientation, and the block is between the end-effector and the goal at the desired goal pose.[1] We perform the same task as in [22] and compare the final results of pushing a squared object on a table surface to two poses (Pose 1 and 2) with a robot arm equipped with a stick. In Table II, we report our findings, with our method showing double the accuracy. Our approach performs continuous pushes, unlike the baseline that stops for replanning after each short push. Thus, we complete either task in approximately 8 seconds, while the baseline takes approximately 4 minutes. We used the same evaluation metric of [22] for the final cost that is a weighted average of position and orientation errors: $1.5(|p_{G_x} - p_{O_x}| + |p_{G_y} - p_{O_y}|) + 0.01|\psi_O - \psi_G|$. For every run, the object is also randomized in the same way as the rollouts. See the accompanying Ł video for the actual behavior.

We further compare our approach with [23] in terms of the success rate of non-prehensile manipulation. Particularly, we consider the same task settings, pushing 5 different objects to 3 different goal poses. To do so we simply change the objects in the simulation and slightly re-tune the MPPI.[2]

Since the trained models from [23] are not provided, we only compare the final results, reported in Table III. Again, thanks to the continuous pushes, our method takes about 3 seconds per task, while the baseline needs about 24 seconds. We performed 10 pushes per object, totaling 150 pushes. For the non-prehensile manipulation task with the robot arm, the mass and friction of manipulated objects have 30% uncertainty, and table friction has 90% uncertainty on the nominal value, sampled uniformly. Size is randomized with zero-mean additive Gaussian noise with a 2 mm standard deviation.

Our method outperforms both baselines in terms of time to completion, accuracy, and success rate, except for one manipulated object. We achieve this without limiting the sampling to 2D end-effector trajectories, without needing learned models, and without requiring inverse kinematics solvers.

2) Extension to Different Robots: Our method is also easily extensible to different robot platforms and objects because it does not require specialized models or controllers that are robot specific, as opposed to the baselines considered. We chose to use an omnidirectional base, and a differential drive robot, to push a box or a sphere to a goal from different initial configurations. To do so, we only need to change the environment and robot URDF in IsaacGym, and re-tune the cost function for pushing due to different hardware.

a) *Omnidirectional push of a box:* The first task is the non-prehensile pushing of a box with an omnidirectional base, see Fig. 7. Success is defined when the box is placed at the goal within 5 cm in the $x - y$ direction and within 0.17 radians in rotation. The robot cannot touch obstacles. The cost function for the MPPI is the same as in (9), re-tuned without considering end effector height and orientation since we now operate on a plane. We add an explicit term for collision avoidance $C_{coll} = \omega_c \sum F_{obst}$:

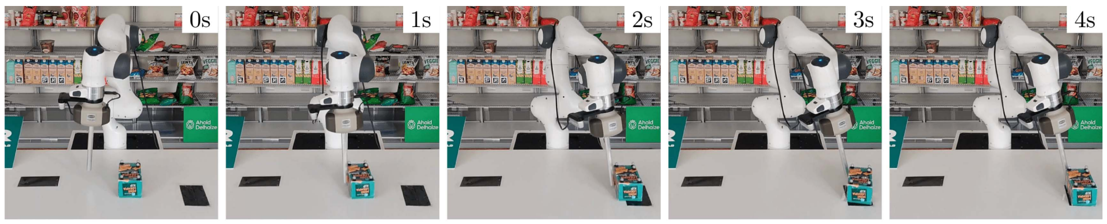
$$C_{push} = C_{dist} + C_{push\ align} + C_{coll}, \tag{11}$$

b) *Omnidirectional push of a sphere:* One can easily extend the example above to different objects with very different dynamics. We chose a sphere instead of a box, and we simply change the object spawned in the simulation. For this task, we want to put the ball in between the two walls, Fig. 8. We considered multiple runs from two different starting poses, A and B. Results are summarized in Table IV and the execution can be seen in the accompanying video.

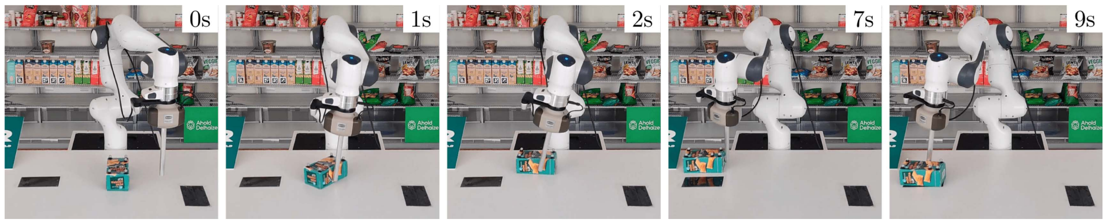
c) *Differential drive non-prehensile pushing:* We perform differential drive non-prehensile pushing, see Fig. 9, with the same

[1]Tuning: $\omega_t = 1, \omega_{O_p} = 16, \omega_{O_r} = 2, \omega_{ee_h} = 8, \omega_{ee_r} = 0.5, \omega_a = 0.8, dt = 0.04, T = 8, K = 500$.

[2]Tuning: $\omega_t = 5$, $\omega_{O_p} = 25$, $\omega_{O_r} = 21$, $\omega_{ee_h} = 30$, $\omega_{ee_r} = 0.3$, $\omega_a = 45$, $dt = 0.04$, $T = 8$, $K = 500$.



(a) Pushing straight to the goal on the right.



(b) Pushing to the goal on the left with 90° rotation.

Fig. 10. Pushing to two goals with a 7 DOF manipulator directly controlling all joint velocities. Our method allows the end-effector to re-position around the object without specifying any desired contact point.

TABLE IV
RESULTS WITH OMNIDIRECTIONAL BASE

Obj	**Env.**	**Runs**	**Time [s]**
Box	Pose A	5	9.66 ± 0.84
	Pose B	5	12.84 ± 0.564
Sphere	Pose A	5	8.76 ± 0.38
	Pose B	5	7.45 ± 0.59

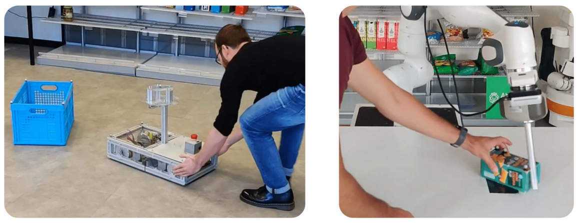


Fig. 11. Qualitative real-world experiments with disturbances. The behavior can be seen in the accompanying video.

cost function as before (see (11)) but re-tuned. One can change the robot for the task by changing the URDF, neglecting all the additional contact modeling required in a classical model-based MPC. The time taken to push the box to the goal was 18.31 s. In the mobile non-prehensile pushing experiments, objects to manipulate are spawned with 30% uncertainty on mass and friction sampled uniformly, while object size is randomized with Gaussian noise with a standard deviation of 5 mm.

D. Real-World Experiments

To demonstrate the applicability of our approach, we transfer to the real world a subset of the non-prehensile manipulation tasks previously presented in Section III-C with both the robot manipulator and the omnidirectional base. In particular, in Fig. 10, we show the results of the 7 DOF manipulator pushing a product to two different goals, similar to the simulations corresponding to Table II. As presented in Fig. 1, the samples are rolled out in $K = 500$ simulated environments in IsaacGym, which, at each timestep, are initialized to the state of the real world. Based on this, the optimal control is estimated and applied to the real system.

When transferring to the real world, compared to the experiments in Section III-C1, only the cost function weights were re-tuned. The horizon, control frequency, number of samples, structure of the cost function, and randomization of the sampled environments remained unchanged.

From the experimental evaluation on the real robot, we observe that the time to complete the pushing tasks and the final position errors are comparable to the results in the simulation from Table II. Importantly, these results are achieved without making assumptions on specific contact points. Thus, the robot can naturally re-position itself and change contact location autonomously. Additionally, our method allows us to sample joint velocities directly; thus, we do not restrict the sampling to 2D end-effector trajectories to be translated into joint commands, as often seen in other approaches.

Lastly, to demonstrate robustness, we disturb the execution of pushing tasks by hand with the manipulator and the omnidirectional base (Fig. 11). Since we do not assume the robot to be behind the object to be pushed for successful execution, and since the planning and execution happen in real-time at 25 Hz, we can largely perturb the task and let the robot compensate.

IV. DISCUSSION

In this section, we discuss key aspects and potential future work related to our solution. First, the computational demands of planning and control with our method can be high when extending the time horizon to several seconds. To keep the time horizon limited for real-time control while preventing being trapped in local minima, future work should incorporate global planning techniques such as A*, RRT, and Probabilistic Roadmaps (PRM) [27] to guide the local planner. Similarly to warm starting predictive controllers [5], [6], one could make use of motion libraries of previous executions or learned policies along with random rollouts, to improve the sampling efficiency and exploration [28]. Second, in real-world scenarios, uncertainties and discrepancies between simulated and actual environments could present challenges for achieving precise movements and manipulation. We utilized randomization of object properties in the rollouts to address some uncertainties. However, online system identification to converge to the true model parameters is not performed. Enhancing the robustness of the MPPI algorithm itself by reducing model uncertainty, as demonstrated by [18], could further improve performance. Third, tuning control algorithms for optimal performance is time-consuming. Implementing autotuning techniques can automate the process and reduce manual effort. Finally, incorporating additional sensor support, such as lidars and signed distance fields, could be beneficial.

V. Conclusion

We presented a way to perform Model Predictive Path Integral controller (MPPI) that uses a physics simulator as the dynamic model. By leveraging the GPU-parallelizable IsaacGym simulator for parallel sampling of forward trajectories, we have eliminated the need for explicit encoding of robot dynamics, contacts, and rigid-body interactions for MPPI. This makes our method easily adaptable to different objects and robots for a wide range of contact-rich motion-planning tasks. Through a series of simulations and real-world experiments, we have demonstrated the effectiveness of this approach in various scenarios, including motion planning with collision avoidance, non-prehensile manipulation, and whole-body control. We showed how our method can compete with state-of-the-art motion planners in case of no interactions, and how it outperforms by a margin other approaches for contact-rich tasks. In addition, we provided an open-source implementation that can be used to reproduce the presented results, and that can be adapted to new tasks and robots.

References

[1] O. M. Andrychowicz et al., "Learning dexterous in-hand manipulation," *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, Jan. 2020.

[2] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: An engineering perspective," *Int. J. Adv. Manuf. Technol.*, vol. 117, no. 5, pp. 1327–1349, Nov. 2021.

[3] F. R. Hogan and A. Rodriguez, "Reactive planar non-prehensile manipulation with hybrid model predictive control," *Int. J. Robot. Res.*, vol. 39, no. 7, pp. 755–773, Jun. 2020.

[4] J. Moura, T. Stouraitis, and S. Vijayakumar, "Non-prehensile planar manipulation via trajectory optimization with complementarity constraints," in *Proc. 2022 Int. Conf. Robot. Automat.*. Philadelphia, PA, USA, May 2022, pp. 970–976.

[5] N. Mansard, A. DelPrete, M. Geisert, S. Tonneau, and O. Stasse, "Using a memory of motion to efficiently warm-start a nonlinear predictive controller," in *Proc. IEEE Int. Conf. Robot. Automat.*, Brisbane, 2018, pp. 2986–2993.

[6] T. S. Lembono, A. Paolillo, E. Pignat, and S. Calinon, "Memory of motion for warm-starting trajectory optimization," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 2594–2601, Apr. 2020.

[7] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *J. Guid., Control, Dyn.*, vol. 40, no. 2, pp. 344–357, Feb. 2017.

[8] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-theoretic model predictive control: Theory and applications to autonomous driving," *IEEE Trans. Robot.*, vol. 34, no. 6, pp. 1603–1622, Dec. 2018.

[9] V. Makoviychuk et al., "Isaac Gym: High performance GPU based physics simulation for robot learning," in *Proc. Neural Inf. Process. Syst. Track Datasets Benchmarks*, J. Vanschoren and S. Yeung, vol. 1, 2021.

[10] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed., Ser. Intelligent Robotics and Autonomous Agents. Cambridge, U.K.: MIT Press, 2011.

[11] A. Li, M. Mukadam, M. Egerstedt, and B. Boots, "Multi-objective policy generation for multi-robot systems using riemannian motion policies," in *Robotics Research*, ser. Springer Proceedings in Advanced Robotics, T. Asfour, E. Yoshida, J. Park, H. Christensen, and O. Khatib, Eds. Cham, Switzerland: Springer, 2022, pp. 258–274.

[12] N. D. Ratliff, K. Van Wyk, M. Xie, A. Li, and M. A. Rana, "Generalized nonlinear and finsler geometry for robotics," in *Proc. IEEE Int. Conf. Robot. Automat.*. Xi'an, China, 2021, pp. 10206–10212.

[13] M. Spahn, M. Wisse, and J. Alonso-Mora, "Dynamic optimization fabrics for motion generation," *IEEE Trans. Robot.*, vol. 39, no. 4, pp. 2684–2699, Aug. 2023.

[14] G. Buizza Avanzini, A. M. Zanchettin, and P. Rocco, "Constrained model predictive control for mobile robotic manipulators," *Robotica*, vol. 36, no. 1, pp. 19–38, Jan. 2018.

[15] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annu. Rev. Control Robot., Auton. Syst.*, vol. 3, no. 1, pp. 269–296, May 2020.

[16] M. Bhardwaj et al., "STORM: An integrated framework for fast joint-space model-predictive control for reactive manipulation," in *Proc. 5th Conf. Robot Learn.*, Jan. 2022, pp. 750–759.

[17] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox, "Object rearrangement using learned implicit collision functions," in *Proc. IEEE Int. Conf. Robot. Automat.*. Xi'an, China, 2021, pp. 6010–6017.

[18] I. Abraham, A. Handa, N. Ratliff, K. Lowrey, T. D. Murphey, and D. Fox, "Model-based generalization under parameter uncertainty using path integral control," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 2864–2871, Apr. 2020.

[19] J. Carius, R. Ranftl, F. Farshidian, and M. Hutter, "Constrained stochastic optimal control with learned importance sampling: A path integral approach," *Int. J. Robot. Res.*, vol. 41, no. 2, pp. 189–209, Feb. 2022.

[20] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, "Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo," 2022, *arXiv:2212.00541*.

[21] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*. Vilamoura-Algarve, Portugal:, 2012, pp. 5026–5033.

[22] E. Arruda, M. J. Mathew, M. Kopicki, M. Mistry, M. Azad, and J. L. Wyatt, "Uncertainty averse pushing with model predictive path integral control," in *Proc. IEEE-RAS 17th Int. Conf. Humanoid Robot.*, 2017, pp. 497–502.

[23] L. Cong, M. Grner, P. Ruppel, H. Liang, N. Hendrich, and J. Zhang, "Self-adapting recurrent models for object pushing from learning in simulation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*. Las Vegas, NV, USA, 2020, pp. 5304–5310.

[24] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, "FORCES NLP: An efficient implementation of interior-point methods for multistage nonlinear nonconvex programs," *Int. J. Control*, vol. 93, no. 1, pp. 13–29, Jan. 2020.

[25] M. Spahn, C. Salmi, and J. Alonso-Mora, "Local planner bench: Benchmarking for local motion planning," 2022, *arXiv:2210.06033*.

[26] G. Williams et al., "Information theoretic MPC for model-based reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 1714–1721.

[27] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[28] E. Trevisan and J. Alonso-Mora, "Biased-MPPI: Informing sampling-based model predictive control by fusing ancillary controllers," *IEEE Robot. Automat. Lett.*, vol. 9, no. 6, pp. 5871–5878, Jun. 2024.