Sampling-Based MPC Using a GPU-parallelizable Physics Simulator as Dynamic Model: an Open Source Implementation with IsaacGym

Corrado Pezzato*, Chadi Salmi*, Elia Trevisan*, Javier Alonso-Mora, Carlos Hernández Corbato

***Abstract*— We present a method for solving finite horizon optimal control problems using a generic physics simulator as the dynamical model. In particular, we present an open-source implementation of a model predictive path integral controller (MPPI), that uses the GPU-parallelizable IsaacGym simulator as the dynamical model to compute the forward dynamics of the system. This allows one to effortlessly solve complex contact-rich tasks such as for example, non-prehensile manipulation of a variety of objects, or picking with a mobile manipulator. Since there is no explicit dynamic modeling required from a user, the repository is easily extendable to different objects and robots, as we show in the experiments section. This makes this method a powerful and accessible tool to solve a large variety of contact-rich tasks. The code available at https://github.com/tud-airlab/mppi-isaac [REPO WILL BE MADE PUBLIC UPON ACCEPTANCE]**

I. INTRODUCTION AND RELATED WORKS

As robots become increasingly integrated into our daily lives, their ability to interact with the environment is becoming more important than ever. From moving obstacles out of the way to picking up objects, robots must be able to plan their motions while accounting for contact with their surroundings. Model Predictive Control (MPC) is a popular approach for robot motion planning. However, contact-rich manipulation poses a challenge due to its non-smooth and hybrid nature, such as transitioning from sticking to sliding motion or entering a contact. Given that most MPC algorithms are based on constrained optimization techniques that assume smooth dynamics to solve the motion planning problem, one has to come up with clever ways to solve the optimization in real-time and rely on extensive modeling to complete an otherwise simple pushing task [1], [2]. In contrast, Model Predictive Path Integral (MPPI) control [3] and its information-theoretic counterpart [4] are sampling-based MPCs, a class of algorithms that rely on parallel sampling of input sequences to approximate the optimal control. This characteristic makes these algorithms gradient-free and therefore suitable for systems with non-linear discontinuous dynamics and cost functions. MPPI has been shown to be capable of controlling a high-degrees of freedom manipulator in real-time [5], albeit still requiring extensive modeling. In recent work [6], a simple sampling-based MPC has been proposed which relies on a physics engine, namely MuJoCo [7], as a dynamic model of the environment for rolling out the sampled input sequences. By doing so, the motion planner offloads all of the modeling efforts to the physics engine, vastly simplifying the controller design. Despite its fast computation, MuJoCo's parallelization capabilities are bounded by the number of threads of the CPU, which limits the number of environments one can sample in parallel to achieve real-time performances.

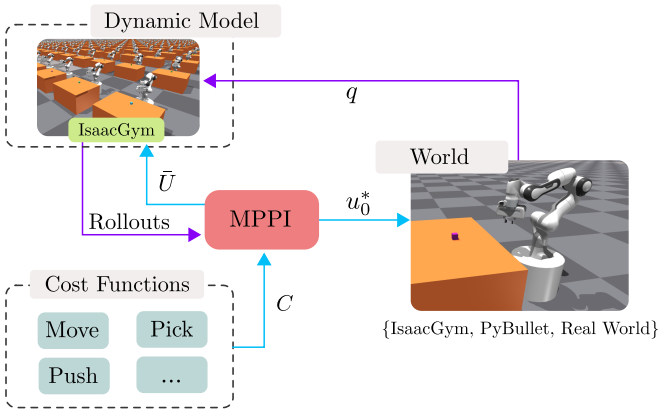

Fig. 1. Scheme of the proposed approach, using IsaacGym as the dynamic model for MPPI. At each time step, IsaacGym is reset to the current world's state q, and random input sequences $\bar{U}$ are applied for the horizon T, to every environment. The resulting rolled out trajectories from IsaacGym are used by the MPPI to approximate the optimal control u_0^* given a cost C.

In this paper, we present an open-source implementation of MPPI that uses IsaacGym [8] as a dynamic model of the environment. Thanks to its GPU parallelization we can take hundreds of samples, thus allowing for real-time control of high-degrees of freedom systems such as a mobile manipulator in contact-rich tasks. We explain the algorithm and the code structure. Then, we show how one can use this method for non-prehensile manipulation and whole-body control for mobile manipulation picking, demonstrating how easy it is to extend to new tasks and robots.

II. METHOD AND SOFTWARE

A. Algorithm

In this section, we give an overview of our implementation of MPPI. For more theoretical insights, please refer to the original publications [3], [4]. Our approach, outlined in Algorithm 1, starts by initializing a previous input sequence U_{prev} to a vector of zeroes of length T, where T is the time horizon in steps. We then sample K sequences of additive input noise $\mathcal{E}_k$ which we will use to explore the input space around U_{prev}. Instead of sampling randomly from a

* Equal contribution, in alphabetical order.

This research was supported by Ahold Delhaize and the "Sustainable Transportation and Logistics over Water: Electrification, Automation and Optimization (TRiLOGy)" project of the Netherlands Organization for Scientific Research (NWO), domain Science (ENW).

The authors are with the Cognitive Robotics Department, TU Delft, 2628 CD Delft, The Netherlands {c.pezzato, c.salmi, e.trevisan, j.alonsomora, c.h.corbato }@tudelft.nl

Gaussian distribution, we sample *Halton Splines* [5]. Once the task begins, we initialize our K simulation environments on IsaacGym to the current observed *world* state q, where the *world* can either be another simulation on IsaacGym or PyBullet, or the real world. In parallel, we can now roll out the sampled input sequences $\bar{U}_k$ into state trajectories Q_k using K simulation environments on IsaacGym and compute their corresponding cost C_k using the desired *objective* function. The cost is discounted over the planning horizon T by a factor γ [5]. To improve numerical stability, the minimum sampled cost is subtracted from each C_k, so that there is always at least one trajectory of cost zero. Next, we can compute the *importance sampling* weights w_k through an inverse exponential of the cost C_k normalized by η, so that $\sum_{k=1}^{K} w_k = 1$. The normalization factor η is a useful metric to monitor, as it gives an idea of how many samples are assigned significant weights. We then update the parameter β, also known as *inverse temperature*, to maintain η between an upper and lower bound. An approximation of the optimal control sequence U^* can now be computed via a weighted average of the sampled inputs. U_{prev} is now updated with U^*, time-shifted backward of one timestep so that it can be used as a warm-start for the next iteration. Out of the entire sequence U^*, only the first input u_0^* is applied to the system, and the next iteration can start.

Algorithm 1 Proposed Approach

Initialize:
  $U_{prev} = [0, \dots, 0]$ ▷ $U_{prev} \in \mathbb{R}^T$
  $\mathcal{E}_k \leftarrow$ *sampleHaltonSplines*() ▷ $k = 1 \dots K$
while *taskNotDone* **do**
  $q \leftarrow$ *observeEnvironment*()
  initializeSimulations(q)
  for $k = 1 \dots K$ **do** ▷ in parallel
    $\bar{U}_k = U_{prev} + \mathcal{E}_k$
    $[Q_k, C_k] \leftarrow$ *computeRolloutCost*($\bar{U}_k, \gamma$)
    $C_k = C_k -$ *minSampled*(C_k)
    $w_k = 1/\eta \exp\left(-(1/\beta) C_k\right)$ ▷ η normalization
  $\beta \leftarrow$ *updateBeta*(β, η)
  $U^* = \sum_{k=1}^{K} w_k \bar{U}_k$
  $U_{prev} \leftarrow$ *timeShift*(U^*)
  applyInput(u_0^*)

B. Implementation

In this subsection, we describe the main components of our open-source implementation, designed to be modular and flexible.

1) MPPI Class: The MPPI class implements the MPPI algorithm and has access to the handle of a function called `dynamics`. This class is agnostic to this function: By default, IsaacGym is responsible for computing the forward dynamics, however, one can also define other custom analytical models for simple problems.

2) IsaacGym Wrapper Class: An IsaacGym wrapper class facilitates interaction with the IsaacGym. This class contains helpful methods for instance to initialize the parallel environments from a list of *configurations* of *actors*. Actors can be either robots or other assets such as boxes or spheres. The wrapper class ensures the separation of concerns between IsaacGym and MPPI.

3) Objective class: Our implementation includes a separate `objective` class that defines how to calculate the cost to be used in MPPI. This class has access to a handle of the IsaacGym wrapper class to calculate the cost. It is therefore straightforward to implement custom costs based on e.g. robot and obstacle states or even contact forces.

4) MPPIisaacPlanner class: Finally, the MPPIisaacPlanner class initializes instances of all the aforementioned classes and ensures that the handle of the `dynamics` and the `objective` function is passed to the MPPI class. This modular approach is flexible and it allows to include different cost functions, robot models, and simulation environments. Through the `compute_action` function, the control action u_0^* is computed based on the state of the world q that is passed as a parameter. Thanks to a dedicated ROS wrapper, one can also forward the commands to real robots.

C. User Guide

To use our method, the user has to provide some configuration files, an objective function, and define a world, see Figure 2.

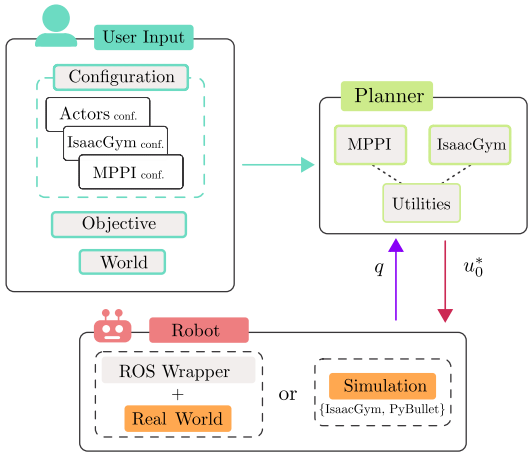

Fig. 2. Simplified system overview from a user perspective. A user has to provide the `Configuration` files for 1) the actors (robot types, URDF, etc.), 2) IsaacGym (integration steps, etc.), and 3) the MPPI (robot-specific tuning and parameters). Then, the user specifies an `objective` to be minimized and the `world`, i.e. the environment and robot for the task.

1) `Configuration` files: Configuration files are required for the actors in the environment, the IsaacGym simulation, and the MPPI planner. These files must specify the relevant parameters for each component, such as the number of rollouts, the MPPI tuning, and the robot's URDF file. Our repository provides example configuration files that one can use as a starting point for different applications.

2) `Objective` class and `world`: The objective defines how to calculate the cost of the rollouts for the MPPI planner in each environment. This involves writing a `compute_cost` function that takes as input a handle

to the instance of IsaacGymWrapper and returns a scalar cost for every environment. The world specifies what objects and robots to populate the environment with, based on the configuration files.

Once provided the `configuration`, `objective`, and `world`, the MPPIisaacPlanner class takes care of setting up the necessary connections between the different components. The resulting object from the MPPIisaacPlanner class allows you to generate control action for your robot.

Overall, this results in a flexible approach where one can adapt the MPPI planner to their own robotic system, without having to worry about the details of the underlying physics simulation or the control algorithm. With our open-source implementation and example files, one can quickly get started with solving complex control tasks in a variety of environments.

III. EXPERIMENTS

In this section, we present a number of use cases with two main purposes. First, we showcase how one can use the repository for non-prehensile manipulation (III-A), and how one can extend it to include different objects (III-A.2) and robots (III-A.3). Second, in III-B we show how complex motions such as picking an object with a mobile manipulator emerge naturally from a simple cost function if one samples all the DOF at once, including the gripper. All the simulations and experiments are carried out on an Alienware m15 R2 laptop, with Intel Core i7-9750H, and Nvidia GeForce RTX 2070.

In each experiment run, the *world* simulation is randomized. Starting from nominal physics properties, objects are actually spawned with 30% uncertainty on mass and friction sampled uniformly, while object size is randomized with Gaussian noise with a standard deviation of 5mm for pushing and 1mm for grasping. To demonstrate robustness to model uncertainties, also the parallel simulations, i.e. our dynamic models, are randomised with the same principle. Therefore, every simulation is different from the others, and all simulations are different from the *world* such that there is always model mismatch. In a sense, we perform a sort of domain randomization in real-time.

Collision avoidance is achieved by heavily penalizing the contact forces between rigid bodies. IsaacGym provides these contacts directly, thus a robot can perform collision avoidance with arbitrarily complex shapes without assuming convex constraints. Additionally, by lowering collision costs, one can allow for contact with a reasonably small exerted force. The simulations and a real-world execution are available in the accompanying video[1].

A. Non-prehensile mobile pushing

1) Omnidirectional base: The first task is the non-prehensile pushing of a box with an omnidirectional base, see Figure 3. Success is defined when the box is placed at the goal within 5cm in the $x-y$ direction and within 0.17 radians in rotation. The robot cannot touch obstacles.

[1] https://youtu.be/I8YKkPTIIZA

We designed the following cost function:

$$C_{push} = C_{dist} + C_{align} + C_{coll}, \tag{1}$$

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

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

The cost C_{align} is an incentive for the robot to keep the object between itself and the goal. We achieve this by considering $cos(\theta)+1$, with θ being the angle between *robot-object* and *object-goal* vectors:

$$C_{align} = \omega_a \left(\frac{\sum (p_R - p_O)(p_G - p_O)}{||p_R - p_O||||p_G - p_O||} + 1 \right)$$

Finally, in the collision cost C_{coll} we consider the contact forces C_F exerted on the obstacles:

$$C_{coll} = \omega_c \sum C_{F_{obst}}$$

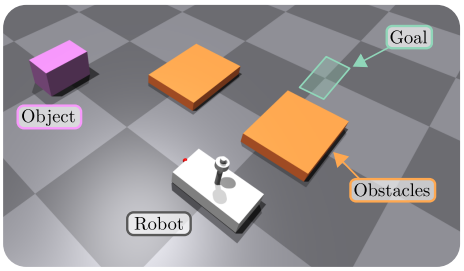

Fig. 3. Non-prehensile task using an omnidirectional base. $\omega_t = 0.2$, $\omega_{O_p} = 2$, $\omega_{O_r} = 3$, $\omega_a = 0.6$, $\omega_c = 10$, $T = 8$, $dt = 0.04$, $K = 300$

2) Extension to other objects: 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, Figure 4.

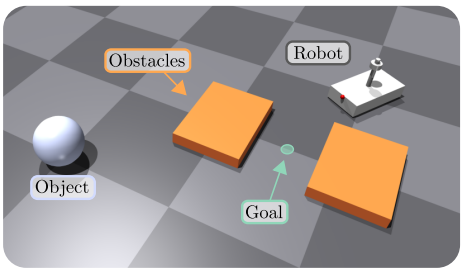

Fig. 4. Non-prehensile pushing of a rolling ball. The goal is to place the ball in between the two obstacles. $\omega_t = 0.2$, $\omega_{O_p} = 0.1$, $\omega_{O_r} = 0$, $\omega_a = 0.1$, $\omega_c = 0.001$, $T = 8$, $dt = 0.04$, $K = 300$

The cost function is the same, with a slightly different tuning since there is no orientation goal, and contacts of

TABLE I

RESULTS OF OMNIDIRECTIONAL BASE NON-PREHENSILE PUSHING

Obj	Env.	Runs	Time [s]
Box	Pose 1	5	9.66 ± 0.84
	Pose 2	5	12.84 ± 0.564
Sphere	Pose 1	5	8.76 ± 0.38
	Pose 2	5	7.45 ± 0.59

small entities are allowed in order to complete the task. Results are reported in Table I for a total of 20 runs, with Pose 1 and 2 being different problem configurations. In every run, objects are randomized.

3) Extension to other robots: Finally, one can also change the robot for the task by simply changing the URDF, neglecting all the additional contact modeling that would be required in a classical model-based MPC. We perform differential drive non-prehensile pushing with the same cost function but re-tuned, see Figure 5. The time taken to push the box to the goal was 18.31s.

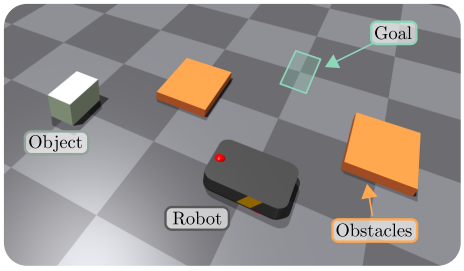

Fig. 5. Non-prehensile pushing with differential drive. The task is the same as for the omnidirectional base. $\omega_t = 0.1$, $\omega_{O_p} = 2$, $\omega_{O_r} = 3$, $\omega_a = 0.6$, $\omega_c = 100$, $T = 12$, $dt = 0.04$, $K = 400$

B. Whole-body control

Our approach scales well with the complexity of the robot. In Figure 6, the task is to relocate an object from a table to an $[x, y, z]$ location using a mobile manipulator with 12 DOF.

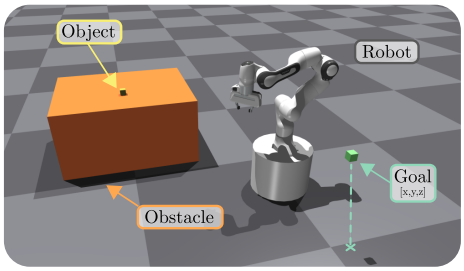

Fig. 6. Whole body motions with a mobile manipulator to bring the cube from an initial location to a final desired one. $\omega_t = 8$, $\omega_{O_p} = 2$, $\omega_{ee} = 1$, $\omega_c = 0.1$, $T = 6$, $dt = 0.04$, $K = 500$

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

$$C_{pick} = C_{dist} + C_{ee} + C_{coll}, \tag{2}$$

where we only consider the Euclidean distance of the end-effector to the object, and the object to the goal:

$$C_{dist} = \omega_t ||p_{EE} - p_O|| + \omega_{O_p} ||p_G - p_O||,$$

and we give the incentive to keep the end-effector pointing downwards, using pitch θ and roll ϕ, with $C_{ee} = \omega_{ee} ||[\phi, \theta] - [0, 0]||$. 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 5 pick-and-deliver tasks, and the time taken was 8.19 ± 1.51s.

We noticed that for smooth whole-body motions of high DOF systems, like this one, one requires a high number of samples (i.e. a few hundred). Empirically, when the number of samples is greater than 50, a GPU pipeline is computationally cheaper than a CPU and scales better. By using IsaacGym, we can compute all the 500 samples required for mobile manipulation in parallel, achieving $25Hz$ real-time control.

IV. CONCLUSIONS

In this paper, we presented an open-source implementation of a sampling-based MPC which uses a parallelizable physics simulator, IsaacGym, as the dynamic model for the rollouts. Using this simple but powerful idea, we showed how one could solve complex contact rich tasks such as non-prehensile pushing and whole-body mobile manipulation for picking. This happens in real-time, without learning, and without requiring additional task-specific dynamical modeling, thanks to the general physics simulator used. Finally, we showed how one can extend the code repository with new robots to perform new tasks with new objects, making this method available to a plurality of researchers. Future work will focus on more real-world experiments, with additional benchmarks against more established baselines for motion planning, non-prehensile manipulation, and whole-body control. Auto-tuning and the addition of global planning to avoid local minima would make this method even more robust.

REFERENCES

[1] F. R. Hogan and A. Rodriguez, "Reactive planar non-prehensile manipulation with hybrid model predictive control," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 755–773, Jun. 2020, publisher: SAGE Publications Ltd STM.

[2] J. Moura, T. Stouraitis, and S. Vijayakumar, "Non-prehensile Planar Manipulation via Trajectory Optimization with Complementarity Constraints," in *2022 International Conference on Robotics and Automation (ICRA)*, May 2022, pp. 970–976.

[3] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, Feb. 2017, publisher: American Institute of Aeronautics and Astronautics Inc.

[4] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and I. A. Theodorou, "Information-Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, Dec. 2018.

[5] M. Bhardwaj, B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos, and B. Boots, "STORM: An Integrated Framework for Fast Joint-Space Model-Predictive Control for Reactive Manipulation," in *5th Annual Conference on Robot Learning*, Jun. 2021.

[6] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, "Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo," Dec. 2022, number: arXiv:2212.00541 arXiv:2212.00541 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2212.00541

[7] 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*, Oct. 2012, pp. 5026–5033, iSSN: 2153-0866.

[8] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac Gym: High Performance GPU Based Physics Simulation For Robot Learning," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, Nov. 2021.