Parallel Monte Carlo Tree Search with Batched Rigid-body Simulations for Speeding up Long-Horizon Episodic Robot Planning

Baihuan Huang  Abdeslam Boularias  Jingjin Yu

Abstract—We propose a novel Parallel Monte Carlo tree search with Batched Simulations (PMBS) algorithm for accelerating long-horizon, episodic robotic planning tasks. Monte Carlo tree search (MCTS) is an effective heuristic search algorithm for solving episodic decision-making problems whose underlying search spaces are expansive. Leveraging a GPU-based large-scale simulator, PMBS introduces massive parallelism into MCTS for solving planning tasks through the batched execution of a large number of concurrent simulations, which allows for more efficient and accurate evaluations of the expected cost-to-go over large action spaces. When applied to the challenging manipulation tasks of object retrieval from clutter, PMBS achieves a speedup of over $30 \times$ with an improved solution quality, in comparison to a serial MCTS implementation. We show that PMBS can be directly applied to real robot hardware with negligible sim-to-real differences. Supplementary material, including video, can be found at https://github.com/arc-l/pmbs.

I. INTRODUCTION

The past decade has witnessed dramatic leaps in robot motion planning for solving problems that involve sophisticated interaction between the robot and its environment, with milestones including teaching quadrupeds to perform impressive tricks [1], [2] and navigate challenging terrains [3], enabling high-DOF robot hands to solve the Rubik’s cube [4], and so on. Whereas some of the success can be attributed to the rapid advancement in deep learning [5] and deep reinforcement learning [6], another undeniable factor is the availability of fast high-fidelity physics engines, including PyBullet [7] and MoJuCo [8]. These physics engines allow the simulation of the physics of complex rigid-body systems, sometimes faster than real-time, which enables the collection of large amounts of realistic system behavior data without even touching the actual robot hardware. Nevertheless, most physics simulators are CPU-based, which can only simulate a limited number of robots simultaneously; this has led to some studies seeking parallelism by using a massive amount of computing resources. For example, the OpenAI hand study [4] used a total of 6,144 CPU cores to train their model for over 40 hours, which is costly and time-consuming.

As physics simulation starts to become a bottleneck in solving robotic tasks, GPU-based physics engines have recently begun to emerge, including Isaac Gym [9] and Brax [10], to address the issue by enabling large-scale rigid body simulation. Early results are fairly promising; for example, the training of the OpenAI hand using Isaac Gym can be done on a single GPU in one hour, translating to a combined resource-time saving of several magnitudes. Similar success has also been realized in applying reinforcement learning on quadrupeds, robotic arms, and so on [9].

In this work, we exploit the power of large-scale rigid body simulation for optimally solving long-horizon episodic robotic planning tasks, such as multi-step object retrieval from clutter, leveraging the strength of another powerful tool that has attracted a great deal of attention – Monte Carlo tree search (MCTS) [11]. MCTS demonstrates clear advantages in solving long-horizon optimization problems without the need of significant domain knowledge [12], and was already employed for solving challenging manipulation tasks [13], [14]. However, even with significant guidance using domain knowledge [14], MCTS incurs fairly long planning times due to its need of carrying out numerous rounds of sequential selection-expansion-simulation-backpropagation cycles. The long planning time, sometimes several minutes per decision

Fig. 1: (a) The hardware setup includes a Universal Robots UR-5e with a Robotiq 2F-85 two-finger gripper and an Intel RealSense D455 RGB-D camera. (b) Planning and simulation carried in physics simulator where thousands of virtual robots operate in parallel. (c) Overview of our system; the small blue cylinder at the center is the target object to be retrieved.

B. Huang, G. Teng, A. Boularias, and J. Yu are with the Department of Computer Science, Rutgers, the State University of New Jersey, USA. Emails: {baichuan.huang, teng.guo, abdeslam.boularias, jingjin.yu}@rutgers.edu. This work is supported by NSF awards 1734492, 1846045, 1845888, and 2132972.
step, limits the applicability of the methods from [13], [14] toward real-time decision making.

Through combining MCTS and large-scale rigid-body simulation with Isaac Gym [9], and carefully introducing parallelism into the mix, we have developed a new line of parallel MCTS algorithms for efficiently solving long-horizon episodic robotic planning tasks. The development of the large-scale rigid-body simulation enabled parallel MCTS is the key contribution of this research, which is highly non-trivial. This is because MCTS has an inherently serial characteristic: as will be explained in more detail, the selection phase of an MCTS iteration depends on the completion of the previous selection-expansion-simulation-backpropagation iteration. Fusing MCTS and Isaac Gym for solving long-horizon manipulation planning tasks also brings significant technical integration challenges because many computational bottlenecks must be addressed for the parallel MCTS implementation to be efficient.

We call our algorithm Parallel Monte Carlo tree search with Batched Simulation (PMBS). As its name suggests, PMBS realizes parallel MCTS computation through batched rigid-body simulation enabled by Isaac Gym. Efficiently combining MCTS and Isaac Gym, PMBS achieves over $30 \times$ speedups in planning efficiency for solving the task of object retrieval in clutter, while still achieving better solution quality, as compared to an optimized serial MCTS implementation, using identical computing hardware. PMBS drops the single-step decision making time to a few seconds on average, which is close to being able to solve the task in real-time. We further demonstrate that PMBS can be directly applied to real robot hardware with negligible sim-to-real differences.

II. RELATED WORK

Task and Motion Planning. Our investigation of long-horizon episodic planning tasks falls under the general umbrella of task and motion planning (TAMP) [15]–[18]. A characteristic that differentiates TAMP and other episodic tasks, e.g., playing rule-based games [12], is the inherently uncountable infinite decision space induced by physical interactions (e.g., pushing, grasping, and so on). The vast search space suggests that the injection of domain knowledge is likely necessary to enable effective planning, e.g., using a combination of symbolic reasoning and sampling-based motion planning techniques [19], [20]. More recently, data-driven methods have also seen increased application to solving TAMP, e.g., using learning to guide the search process [21], [22], predicting the outcome of actions [23]–[25], directly predicting feasibility [26], combining neuro-symbolic task planning and motion primitives [27], and so on.

Object Retrieval. Object retrieval from clutter, the long-horizon task that we focus on in this study, can be viewed as a form of rearrangement planning [28], [29]. Online planning for object search with partial observations has been discussed in [30]. Retrieving objects under occlusion was also recently considered in [31] where parallel-jaw and suction grasping were used along with pushing to de-clutter surroundings of target objects. A model-free reinforcement learning technique has also been used for searching for objects in [32]. In [33], an agent was trained to find a continuous trajectory of a gripper that pushes away clutter or pushes the target object to free space, mimicking human-like behavior. A human in-the-loop solution was proposed in [34] to help with searching for objects in clutter. A deep Q-Learning method [35] considers a similar task and setup but uses additional primitives such as sliding objects from the top. Our work partially builds on [13], [14], which used MCTS for object retrieval, but with the goal of significantly accelerating the planning process.

Grasping and Singulation. The retrieval task that we tackle is closely related to other manipulation tasks including grasping and singulation. Challenges including friction modeling and inertia estimation has led to the arise of data-driven grasping methods [36], [37]. Recently, grasping in clutter has received more attention [38]–[41]. Convolutional Neural Networks are widely used to construct grasp proposal networks such as Dex-net 4.0 [42], which are trained to detect 6D grasp poses in point clouds [43]. Singulation, i.e., isolating specific object(s) from the rest [44], is necessary for object retrieval. Usually, a sequence of pushing and grasping actions is used to clear the clutter that surrounds the target object. In [45], a model-free method was used to learn a reactive pushing policy without long-horizon reasoning. Later, other model-free reinforcement learning algorithms [46], [47] used learned push policies to improve grasping.

III. PRELIMINARIES

A. Problem Formulation

In this paper, we task a robot equipped with a camera and a two-finger gripper to grasp a desired object from a densely packed clutter, as a concrete instance of long-horizon episodic robot planning problems. The workspace is a confined planar surface. Two types of primitive actions are allowed: pushing and grasping. All objects are rigid; the target object has a different color to facilitate its detection. The only observation available to the robot is an RGB-D image that is taken by a top-down fixed camera, as shown in Fig. 1. Every time the robot executes a push or a grasp action, a new image is taken. A similar problem has been previously defined in [13], [14], [24]. Compared to [13], [14], the problem addressed in the present work is significantly more challenging to solve because the workspace is confined to a substantially smaller area, while keeping the number and sizes of objects the same. Consequently, the free space between the objects is reduced, and the robot needs to find a larger number of shorter surgical push actions in order to free the target object and grasp it. In fact, we found from our experiments (Sec. V) that the original setup considered in [13], [14] can be solved using a brute-force parallel search in a GPU-based physics simulator, without a Monte Carlo tree search.

The object-retrieval-from-clutter task can be formalized as a Markov Decision Process (MDP) defined as $(S, A, T, R, O, \gamma)$, to minimize the number of pushes needed for retrieving the target. $\gamma \in [0, 1]$ is a discount factor. This MDP is described in the following.
State space \((S)\) is a bounded workspace \(W\) containing \(n\) objects. A state \(s_t\) at time \(t\) is defined as \((\text{robot}_t, \text{obj}_1^t, \text{obj}_2^t, \ldots, \text{obj}_n^t)\) wherein \(\text{robot}_t\) is a vector of the robot’s joint angles and \(\text{obj}_i^t\) is a vector of the object \(i\)’s pose and geometry. Action space \((A)\) is the union of two subsets: push actions \(A^p\) and grasp actions \(A^g\). A push \(a^p \in A^p\) is a quasi-static planar motion defined as \(a^p = (x_1, y_1, x_2, y_2)\) where \((x_1, y_1)\) and \((x_2, y_2)\) are the start and end points of the gripper tip in a straight line motion. A grasping action \(a^g \in A^g\) is a parallel jaw grasp defined as \(a^g = (x, y, z, \theta)\), where \((x, y, z)\) is the center point between the gripper’s two fingers, and \(\theta\) is the rotation angle in the \(z\)-axis of the workspace. The opening distance of two fingers is fixed. Transition function \((T)\) maps a state \(s_t\) to \(s_{t+1}\) given action \(a_t\) according to laws of quasi-static motion and friction, while taking into account collisions. Reward function \((R)\) returns a scalar \(r = \{0, 1\}\) given input state \(s_t\). If a target object is grasped, then \(r_t = 1\), otherwise \(r_t = 0\). All push actions are given a zero immediate reward. Observation space \((O)\) contains RGB-D images \(o_t\) taken from the top-down camera.

B. Monte Carlo Tree Search

Monte Carlo tree search (MCTS) [11] was notably employed for playing turn-based games like chess and go. As a result, MCTS is clearly applicable to long-horizon episodic robot planning tasks. MCTS builds a search tree, balancing exploration and exploitation, by iteratively performing selection-expansion-simulation-backpropagation operations. In the selection phase, MCTS selects a best node to grow the tree. A popular node selection criterion is based on the upper confidence bound (UCB) [48], [49],

\[
\arg\max_{n' \in \text{children of } n} \frac{Q(n')}{{N'(n')}} + c \sqrt{\frac{2 \ln N(n)}{N(n')}},
\]

where \(Q(n)\) is the sum of rewards collected starting from the state corresponding to node \(n\), \(N(n)\) is the number of times \(n\) was selected so far. The selection process continues until it finds a node that corresponds to a terminal state or a node that has never-explored children. We note that a node \(n\) is always associated with a state \(s\) and an observation \(o\); sometimes a node \(n\) and the corresponding state \(s\) are used interchangeably.

After a node \(n\) is selected, if it is not a terminal node, it will be expanded and its new child, say \(n'\), will be added to the search tree. Subsequently, a simulation will be carried out at \(n'\). This selection-expansion-simulation process is repeated until a terminal state (or a stopping condition) is reached, which yields a reward. The obtained terminal reward is propagated back from \(n'\) all the way to the root node, while updating the sum of reward \((Q(n))\) and incrementing the number of visits \((N(n))\) for all the nodes along the path.

The entire selection-expansion-simulation-backpropagation procedure is repeated for several rounds, after which MCTS selects the action at the root node that yields the child with the best average reward to execute on the real robot. The operations of MCTS are visualized in Fig. 2 (the upper box).

![Fig. 2: Outline of Monte Carlo tree search as adapted for object retrieval.](image)

IV. METHODOLOGY

Effectively employing MCTS to tackle long-horizon episodic robot planning requires a highly non-trivial adaptation of MCTS. In this section, we first describe the necessary preparation for integrating MCTS and physics simulation for object retrieval, then augmentations to the architecture for GPU-based processing, and then outline our key ideas and design choices in our parallelization effort.

A. Serial MCTS for Object Retrieval from Clutter

To use MCTS for the object retrieval task and solve real instances, we integrate it in a process that alternates between search in simulation and execution on the real system. Our MCTS process takes in a scene that is segmented into objects, and uses physics simulation to reason about the proper push actions to facilitate the final retrieval of the target object. An overview of the MCTS process is provided in Fig. 1. In other words, we first replicate in the simulator the real perceived scene at the beginning of each episode, perform computation and simulation, and then execute with the real robot the action that results from the simulation to guide the resolution of the retrieval task on the real objects.

We now describe the details of our basic serial MCTS adaptation. For the selection step, the standard UCB formula, Eq. (1), is used. For the expansion step, for a selected node \(n\) that has not been expanded, we sample many potential push directions by examining the contour of the objects. These sampled pushes become the candidate actions under \(n\) for expansion. After a sampled push action is chosen, the action is executed in the physics simulation and a new node is added to the tree. The MCTS simulation step is then carried out with additional consecutive random pushes to obtain a reward for the newly added node. Note that for each simulation step, we must decide whether the resulting state is a terminal state; this is done using a grasp classifier, to be explained later.

An important design decision we make here, to render MCTS computation tractable, is to limit the depth of the tree. We limit the depth of the overall tree to be no more than some \(d_T\). The simulation can be carried out for at least \(d_s\) steps. This means that the maximum depth reached by MCTS does not exceed \(d_T + d_s\). If expansion happens at depth \(d_T\), we allow the state to be simulated further until \(d_T + d_s\). Given our goal of finding the least number of pushes...
for retrieving the target object, \( d_T \) and \( d_s \) can potentially be dynamically updated when an identified terminal node has a depth \( d \) smaller than \( d_T \). In this case, we set \( d_T = d \) and \( d_s = 0 \). We terminate an MCTS process if: (1) the elapsed time exceeds a preset budget \( T_{\text{max}} \), (2) the tree is fully explored, or (3) the target can be grasped in an explored node and all nodes at its parent’s level have been explored.

After each full MCTS run, we execute the best action it returns on the actual scene (simulated or real), and then use a grasp network (GN) [13], [14] to tell us whether the target object is retrievable. If it is, GN further tells us how to grasp it; the task is then completed. Details of GN, for replication purposes, can be found in the online supplementary material.

### B. Adaptions for GPU

Besides simulation, which can be sped up using GPU-based physics engines, there are three additional bottlenecks in the process to parallelize MCTS for object retrieval. One of these is the parallelization of MCTS itself and the other two are specific to the object retrieval problem: action sampling and grasp feasibility prediction. We leave the first bottleneck for Sec. IV-C and address the latter two here.

**Speeding up Action Sampling.** Because the number of push action choices is uncountably infinite, action sampling is necessary. We modified the action sampler from [13], [14] with slight changes and a more efficient implementation. As shown in Fig. 3, for a given \( o_t \), actions \( o_{t+1} \) are sampled around the clutter. \( N_a \) actions are evenly sampled around the contour of each object, from edge to center. Actions that cannot be executed due to collisions are discarded. Further speedups are obtained by pre-computing the sampled actions for each object and only performing collision checking between the robot’s start pose of push and objects at runtime.

**Grasp Evaluation.** Previous learning-based methods for object retrieval use a grasp network (GN) to evaluate the feasibility of grasping the target object [13], [14], which becomes a time-consuming bottleneck when parallelized directly. GN is relatively slow because it evaluates all state-action pairs in the selected collection are unique. For this reason, knowing the number of possible grasp poses. However, knowing the best grasp pose is unnecessary if we only want to know how “graspable” a state is. Given this observation, we develop a simplified grasp classifier (GC) that only returns a grasp probability (Fig. 4). GC is an EfficientNet-b0 [50] that takes a depth image as input and outputs a logit between 0 and 1. Given a depth image and a target object, we can query GC whether the target object is graspable by comparing its output to a preset threshold \( R_g \). Details about GC’s implementation and training in simulation can be found in the online supplementary material. Note that GN is still used after each full MCTS run for potentially grasping the target object, as described in Sec. IV-A.

### C. Parallel MCTS with Batched Simulation

Given the availability of powerful GPU-based physics simulators including Isaac Gym [9] and Brax [10], which enables the simulation of a large number of systems independently and simultaneously, a natural route for speeding up long-horizon episodic robot planning tasks is to introduce parallelism into the MCTS pipeline outlined in Sec. IV-A, to perform many simultaneous simulations. However, it is challenging to introduce parallelism into MCTS because optimal node selection depends on the reward of all previous rounds. To enable parallelism in MCTS for object retrieval from clutter and harness GPU-based simulation, we introduce the following modifications to the MCTS procedure outlined in Sec. IV-A. We assume the number of parallel environments in the simulator is fixed to some number \( N_e \). Each parallel environment contains an identical virtual robot and objects.

**Selection with Virtual Loss.** By observing the operations of MCTS, it is not difficult to see that the parallelism of MCTS requires modifying the UCB formula. Otherwise, the same node in a search tree will be selected for expansion in multiple parallel environments, leading to redundancy and poor performance. To address this issue, we borrow the idea of virtual loss [51], which has shown to give good results in multiple application domains [52]–[54]. Virtual loss is used to adjust the calculation of UCB values for the nodes that have been selected but not yet expanded [51], [55],

\[
\arg \max_{n' \in \text{children of } n} \frac{Q(n')}{N(n')} + \frac{2 \ln \left( \frac{N(n)}{N(n')} + \frac{\hat{N}(n)}{\hat{N}(n')} \right)}{N(n')} + c \sqrt{\frac{2 \ln \left( \frac{N(n)}{N(n')} + \frac{\hat{N}(n)}{\hat{N}(n')} \right)}{N(n') + N(n')}},
\]

where the \( \hat{N}(n) \) is the number of selected but not yet expanded nodes under node \( n \). \( \hat{N}(n) \) will be reset to zero once the selection phase of parallel MCTS is completed. Basically, Eq. (2) penalizes selecting nodes that have already been selected in some other parallel environment but for which expansion and simulation have not yet been completed. With Eq. (2), it is still possible for a node \( n \) to be selected multiple times, which may lead to redundant simulations. To avoid this and ensure that no redundant simulations are carried out, we mark all selected actions of node \( n \) and share this information across all the parallel environments.

A collection of state-action pairs is returned from the selection phase. The same state could be selected many times, but all state-action pairs in the selected collection are unique. For example, in Fig. 5, upper left, \( (s^1_{t+2}, a^1) \), \( (s^2_{t+2}, a^1) \), \( (s^3_{t+2}, a^1) \) are four such state-action pairs. Batch-mode expansion/simulation on this collection is then performed in parallel using GPU.

**Batch Expansion.** After a batch of state-action pairs (\( \{(s, a)\} \)) has been selected, the expansion step is carried out for all of these pairs simultaneously. For this purpose, environments in the simulator (Isaac Gym) will be loaded with the appropriately selected states, after which expansion (transition) is carried out in parallel. A batch expansion creates a set of new nodes added to the tree, each of which is different.
In Fig. 5, lower left, \( s_{i+1}^{1}, \ldots, s_{i+2}^{i} \) are the result of expanding 
\( (s_{i+1}^{1}, a^{1}), \ldots, (s_{i+1}^{i}, a^{i}) \), respectively.

**Algorithm 1: Parallel MCTS with Batched Simulation**

1. Function \( \text{Main}(s_i, a_i) \)
2. \( \text{while} \) there is a target object in workspace \( \text{do} \)
3. \( \quad \text{if} \) the target object can be grasped (query GN) \( \text{then} \)
4. \( \quad \quad \) Execute grasp the target object
5. \( \quad \text{else} \) Execute \( \text{Parallel-MCTS}(s_i) \) \( \text{// Push} \)
6. Function \( \text{Parallel-MCTS}(s) \)
7. Create root node \( n_0 \) with state \( s \)
8. \( \text{es_level} \leftarrow 1 \) \( \text{// Early stop level} \)
9. \( \text{graspable_nodes} \leftarrow \emptyset \)
10. \( \text{while} \) (within time budget) \( \text{and} \) (depths of all 
11. \( \quad \text{graspable_nodes} \) are greater than \( \text{es_level} \) \( \text{do} \)
12. \( \quad \quad \) \( ([n^1, a^1], \ldots, [n^N, a^N]) \leftarrow \text{Selection}(n_0) \)
13. \( \quad \quad \) Reset all \( \hat{N}(n) \) to 0
14. \( \quad \quad \) \( [n^1, \ldots, n^N] \leftarrow \text{Expansion}([n^1, a^1], \ldots, [n^N, a^N]) \)
15. \( \quad \quad \text{for} \ n^i \ \text{in} \ n^1, \ldots, n^N \text{do} \)
16. \( \quad \quad \quad \text{if} \ GC(n^i(a)) > \hat{R}_c \text{then} \)
17. \( \quad \quad \quad \quad \text{if} \ \text{all nodes at es_level - 1 are fully expanded or} \)
18. \( \quad \quad \quad \quad \text{terminal then} \)
19. \( \quad \quad \quad \quad \quad \text{es_level} \leftarrow \text{es_level} + 1 \)
20. \( \quad \quad \quad \quad \text{Backpropagation}([n^i, r^1], \ldots, [n^i, r^N]) \)
21. \( \quad \text{return} \) the \( a^i \) that leads to best child node of root, 
ranked by Eq. 2.

**Batch Simulation.** The batch simulation step of our parallel 
MCTS implementation is similar to the batch expansion step, 
but with additional steps inserted before and after. Before a 
push simulation, random actions must first be selected, using 
the action sampling method outlined in Sec. IV-B. After each 
push simulation, GC, as described in Sec. IV-B is applied 
to evaluate the outcome. As we can see, to best exploit 
the parallelism from the simulator, action sampling and GC 
should be carried out as efficiently as possible, so that they 
do not become significant computational bottlenecks.

During simulation, we also perform leaf parallelization 
[51] when the number of simulation environments is more 
than the number of states for which MCTS simulations are to 
be carried out. This is reflected in Fig. 5, upper right, where 
the first two states each are simulated twice initially. If some 
environments, after a push, are predicted by GC as graspable, 
then further simulation on these environments will not be 
carried out, and these environments can be re-purposed. For 
example, in Fig. 5, upper right, a simulation under the second 
state terminates early, and the associated environment can be 
used to perform additional simulation for the first state.

**Backpropagation.** The backpropagation phase is straightforward 
to execute, as it simply backpropagates the rewards to 
the root of the tree. We note that, for a single state for which 
multiple simulations are carried out, it is natural to select the 
maximum reward obtained instead of taking averages (see 
Fig. 5, lower right).

The pseudo-code of PMBS is given in Alg. 1 with the 
selection subroutine given in Alg. 2. Other subroutines of 
PMBS are mostly straightforward.

**Algorithm 2: Selection with Virtual Loss**

1. Function \( \text{Selection}(n_0) \)
2. \( \quad \text{Pairs} \leftarrow \emptyset \)
3. \( \quad \text{while} \) \( \text{len(Pairs)} < N_e, \text{do} \)
4. \( \quad \quad n^i \leftarrow \text{traverse tree until a leaf node using Eq. 2} \)
5. \( \quad \quad a^i \leftarrow \text{one sampled action of } n^i \)
6. \( \quad \quad \text{Remove } a^i \text{ from the sampled actions of } n^i \)
7. \( \quad \quad \text{Pairs} \leftarrow \text{Pairs} \cup \{(n^i, a^i)\} \)
8. \( \quad \quad \hat{N}(n^i) \leftarrow \hat{N}(n^i) + 1 \)
9. \( \quad \quad \text{increment virtual counts of ancestor nodes of } n^i \)
10. \( \quad \text{return} \) Pairs

V. EXPERIMENTAL EVALUATION

We evaluated the proposed system (PMBS) in a physics simulator 
(Isaac Gym) and on a real robot on adversarial 
test cases. In comparisons to baseline and ablation studies, 
we observe significant improvements using the GPU-based 
physics simulator together with parallel MCTS, which brings 
episodic decision making for real robots closer to real-time, 
i.e., a single complex decision is made in a few seconds. All 
experiments are evaluated on a desktop with an Nvidia RTX 
2080 Ti GPU, an Intel i7-9700K CPU, and 32GB of memory.

A. Simulation Studies

In this work, the simulated environment is built with Isaac 
Gym [9], consisting of a Universal Robot UR5e with a two-

finger gripper Robotiq 2F-85, and an Intel RealSense D455 
RGB-D camera overlooking a tabletop workspace as shown 
in Fig. 1. The robot is in position-control mode, push and grasp 
actions are controlling the position of the end-effector, inverse 
kinematic [56], [57] is applied to convert to joint space. The 
effective workspace is at a size of 0.288 × 0.288m, discretized 
as a grid of 144 × 144 cells where each cell is one pixel in 
the image (orthographic projection) taken by the camera. 
The workspace, in comparison to previous studies [13], [14], 
is significantly smaller (only about 45% in terms of area),
making the setting much more challenging. We intentionally selected the setting to demonstrate the power of PMBS.

All objects should reside in the workspace. 20 cases [13] used for evaluation can be found in Fig. 6, where the red lines denote the boundary to which objects centers must be confined at all times. The push distance of a push action is fixed at 5cm (10cm in previous work [13], [14]), the effective push distance is around 3cm (the distance objects are moved).

**Metrics.** Four metrics are used to evaluate our systems: 1) the number of actions used to retrieve the target object, 2) the total planning time used for retrieving the target object (build the tree), 3) the completion rate in retrieving the target object within 16 actions, 4) the grasp success rate, which is the number of successful grasps divided by the total number of grasping attempts.

**Baseline.** We use an optimized serial MCTS implementation as the baseline, where the number of environments used for MCTS is one. The following hyperparameters are used across all methods in benchmark unless otherwise mentioned.

The discount factor $\gamma = 0.8$. The default max depth of tree $d_T = 7$, and the default simulation depth $d_s = 3$. The threshold of GC is $R^*_c = 0.9$. The UCB exploration term $c$ in Eq. 1 and 2 is 0.3. The time limit (budget) $T_{\text{max}}$ for one step planning is 60 seconds. 1000 robots (environments) in Isaac Gym are used in our PMBS; it takes around 2.2 seconds for all robots to complete one push action.

We evaluate the performance of PMBS and the baseline serial MCTS over all 20 cases, running each case five times. For the evaluation, we set a time budget $T_{\text{max}} = 60$ and denoted the two methods as PMBS-60 and MCTS-60, respectively. The summary benchmark for these two methods can be found in the first two rows of Table 1; individual results for each case can be found in Figs. 7 and 8.

We make some observations based on the result. First, PMBS outperforms the serial MCTS version in terms of number of actions and computation time across all cases, which is as expected because PMBS engages many environments to facilitate its search effort. On the other hand, when we view the solution quality and computation time together, the advantage of PMBS over serial MCTS is significant: PMBS-60 uses 35 seconds on average for planning, whereas MCTS-60 uses over 300 seconds. This along translates to an 8.6× speedup. At the same time, PMBS-60 uses 70% fewer actions in solving the tasks. A further data point regarding the speedup at the same solution is given a bit later in Fig. 9.

A second observation is that, despite the fact that we are dealing with a difficult long-horizon planning problem, PMBS is able to achieve planning that is close to being able to perform reasoning in real-time, as it takes an average of 35/3.91 < 9 seconds to make a single decision. With further optimization and/or better hardware, we believe that PMBS will achieve real-time decision-making capability for the current set of object retrieval tasks.

**Ablation Study.** The time budget $T_{\text{max}}$ is one of the main factors that influence the solution quality and planning time. To understand its role, several time budgets are used to evaluate our method, as shown in Fig. 9. Given more time for tree search, serial MCTS and PMBS could improve the solution quality, leading to fewer required actions. The trends of serial MCTS (number of action) are steep, as it is highly possible that it could not find a solution given a limited time. The trend of PMBS (planning time) is more gradual, as the most time-consuming search happens in the first few iterations, which usually uses all the time budget. While serial MCTS never achieves the same solution quality as PMBS, comparing the first PMBS data point and the last serial MCTS data point, we observe a 855/28 = 30× speedup with PMBS still has some quality advantage.

On the flip side, we note that the speedup of 30× seems small considering that we used 1000 environments. This is due to two factors. First, MCTS is itself a serial process; parallelization will incur performance loss. Second, while we have improved many bottlenecks, e.g., on action sampling and grasp classification, the object retrieval task contains many elements that cannot be readily parallelized.

We also evaluated the impact of the exploration and exploit trade-off on PMBS. If the $c$ in Eq. (2) is set to 0, i.e., pure
exploitation, PMBS-60 uses 4.03 actions and 113 seconds (planning time) in average on 20 cases. The performance is worse than when \( c = 0.3 \), as shown in Table. I. This is expected as the greedy approach could be stuck in a local optimum. PMBS-60 is also tested by setting \( c \) to be a large number in Eq. (2), i.e., pure exploration, which uses 12.71 actions and 147 seconds (planning time) on averages; the completion rate has a steep drop to 42.0%.

---

### Additional Experimental Details

Curious readers may find in the online supplementary material additional experimental details including complete, actual execution snapshots of PMBS and MCTS for all 20 cases, as well as the execution snapshots for real robot experiments.

---

### VI. Conclusion

In this work, we propose PMBS, a novel parallel Monte Carlo tree search technique with GPU-enabled batched simulations for accelerating long-horizon, episodic robotic planning tasks. Through carefully making a series of design choices to overcome multiple major bottlenecks resisting the parallelization effort, PMBS achieves over \( 30 \times \) speedups compared to a decent serial MCTS implementation while still delivering better solution quality, using the same computing hardware. Real robot experiments show that PMBS directly transfers from simulation to the real physical world to achieve near real-time planning performance in solving complex long-horizon episodic robot planning tasks.

---

### REFERENCES

[1] X. B. Peng, E. Coumans, T. Zhang, T.-W. E. Lee, J. Tan, and S. Levine, “Learning agile robotic locomotion skills by imitating animals,” in *Robotics: Science and Systems*, 07 2020.

[2] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, “Learning agile and dynamic motor skills for legged robots,” *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[3] A. Kumar, Z. Fu, D. Pathak, and J. Malik, “RMA: Rapid Motor Adaptation for Legged Robots,” in *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

[4] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozeffowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al., “Learning dexterous in-hand manipulation,” *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, vol. 25, 2012.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[7] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning,” http://pybullet.org, 2016–2021.

[8] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for model-based control,” in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.

[9] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al., “Isaac gym: High performance gpu-based physics simulation for robot learning,” *arXiv preprint arXiv:2108.10470*, 2021.

[10] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, “Brax - a differentiable physics engine for large scale rigid body simulation,” 2021. [Online]. Available: http://github.com/google/brax.

[11] R. Coulom, “Efficient selectivity and backup operators in monte-carlo tree search,” in *International conference on computers and games*. Springer, 2006, pp. 72–83.

[12] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., “Mastering chess and shogi by self-play with a general reinforcement learning algorithm,” *arXiv preprint arXiv:1712.01815*, 2017.

[13] B. Huang, S. D. Han, J. Yu, and A. Boularias, “Visual foresight trees for object retrieval from clutter with nonprehensile rearrangement,” *IEEE Robotics and Automation Letters*, vol. 7(1), pp. 231–236, 2022.

[14] B. Huang, T. Guo, A. Boularias, and J. Yu, “Interleaving monte carlo tree search and self-supervised learning for object retrieval in clutter,” in *2022 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.
