Enhancing Dexterity in Robotic Manipulation via Hierarchical Contact Exploration

Xianyi Cheng, Sarvesh Patil, Zeynep Temel, Oliver Kroemer, and Matthew T. Mason

***Abstract*— We present a hierarchical planning framework for dexterous robotic manipulation (HiDex). This framework exploits in-hand and extrinsic dexterity by actively exploring contacts. It generates rigid-body motions and complex contact sequences. Our framework is based on Monte-Carlo Tree Search (MCTS) and has three levels: 1) planning object motions and environment contact modes; 2) planning robot contacts; 3) path evaluation and control optimization that passes the rewards to the upper levels. This framework offers two main advantages. First, it allows efficient global reasoning over high-dimensional complex space created by contacts. It solves a diverse set of manipulation tasks that require dexterity, both intrinsic (using the fingers) and extrinsic (also using the environment), mostly in seconds. Second, our framework allows the incorporation of expert knowledge and customizable setups in task mechanics and models. It requires minor modifications to accommodate different scenarios and robots. Hence, it could provide a flexible and generalizable solution for various manipulation tasks. As examples, we analyze the results on *7* hand configurations and *15* scenarios. We demonstrate *8* of them on two robot platforms.**

I. Introduction

Robots need dexterity to perform daily manipulation and complex industrial tasks. Consider taking a book from the bookshelf. The robot should consider the occlusion of the bookshelf and other books, even use them, to get the book out. The robot need to not only use it own fingers dexterously, but also be smart about exploiting its environment, as "external" fingers to support the movements of the object.

The automatic planning of intrinsic and extrinsic dexterity for general manipulation tasks still remains challenging. First, as contacts introduce discontinuity and changes in system dynamics, planning through contacts is particularly difficult [1], especially considering both robot and environment contacts for the object. Second, due to the diverse nature of manipulation, it is hard to predefine all the possibilities for dexterity as motion primitives. It is important for the robot to be able to discover dexterous motions by its own. Third, current manipulation planners or policies are often tailored for specific problems. A complex in-hand manipulation pipeline [2] cannot directly solve on-table object reorientation problems in [3], and neither of them can be directly applied to planar pushing [4]. As real-world manipulation problems are often mixes of specific manipulation tasks, it is important for a general manipulation planner to cover different tasks.

The authors are with Carnegie Mellon University, Pittsburgh, PA, 15213, USA. {xianyic, sarveshp, ztemel, okroemer, mm3x}@andrew.cmu.edu

Video: https://youtu.be/fScfat1Ys6U

Code: https://github.com/XianyiCheng/HiDex

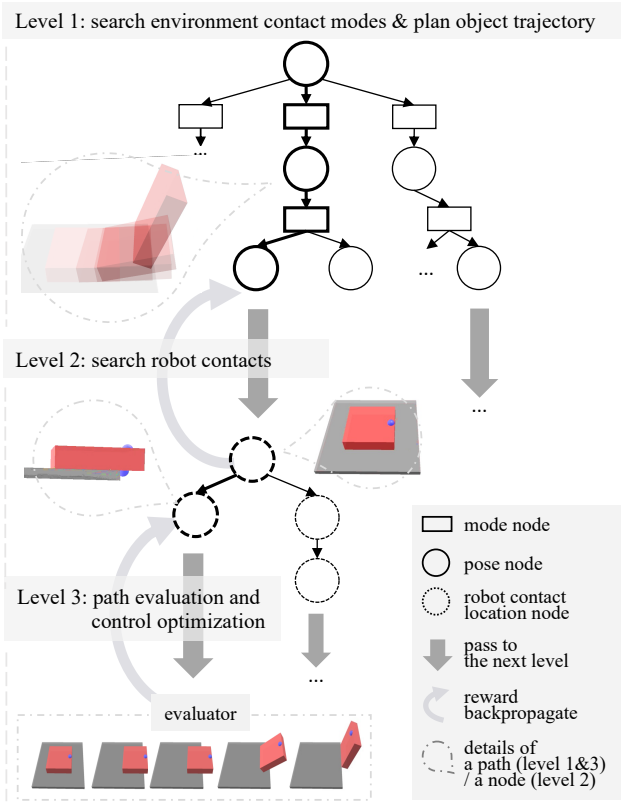

Fig. 1: An overview of our framework, with an example of picking up a card. The following processes run iteratively. Level 1 plans object trajectories, interleaving searches over discrete contact modes (▭ nodes) and continuous object poses (◯ nodes). An object trajectory is passed to Level 2 (⇩) to plan robot contact sequences on the object surface (◌ nodes). The full trajectory of object motions and robot contacts is passed to Level 3 (⇩) for evaluation and control optimization. After evaluation, Level 3 passes the reward back to the upper levels (↻). The reward is updated for every node in the path (bold nodes). In the example, the robot pulls the card the edge of the table and then grasps it.

We propose a *hierarchical* framework, as shown in Figure 1, aiming to address challenges mentioned above. We take an object-centric view to represent the object contact interactions with the environment and the robot. To effectively explore the complex space created by contacts, we exploit a hierarchical structure combined with MCTS[5]. In Level 1, we perform object trajectory and environment contact mode planning. Environment contact information

(contact modes) are used to guide the generation of object motions — which we consider as the active exploration of extrinsic dexterity. In Level 2, given an object trajectory, the intrinsic dexterity is planned by optimizing for robot contact sequences on the object surface. In Level 3, more details and optimization of the plans are computed and rewards are backpropagated. MCTS used in Level 1 and 2 allows us to encode expert knowledge and information gathered during the search process as heuristics, guiding the search directions and balancing exploitation and exploration. In addition, we employ Rapidly-exploring random tree (RRT) [6] as the MCTS rollout to enhance the exploration of object configuration space in Level 1.

Our design works with different task mechanics, robot hand configurations, object and environment models. It is easy to configure many new scenarios with simply one file in our code. It is flexible to encode expert knowledge into the search through MCTS action policies, value estimations, and rewards. We instantiate this framework on manipulation with extrinsic dexterity and in-hand manipulation. Demonstrated tasks include *pick up a card*, *book-out-of-bookshelf*, *peg-out-of-hole*, *block flipping*, *occluded grasp*, *upward peg-in-hole*, *sideway peg-in-hole*, *planar reorientation*, *planar block passing*, and *in-hand reorientation*. As discussed in Section VI, we envision this framework can be extended towards general manipulation planning that incorporates global reasoning, mechanics, learning, and optimization.

II. Related Work

A. Dexterous Manipulation Planning

Most works in manipulation focus on individual skills, like pushing [7][4], pivoting [8], [9], [3], [10], tumbling [11], grasping [12][13], on-table reorientation [3], and predefined dynamic skills [14]. While the mechanics and planning of specific motion types are studied in depth, generating dexterous manipulation planning is still under-explored.

What are the essential challenges about dexterous manipulation planning? The presence of potential contact changes introduce discontinuity and changes in system dynamics (non-differentiable). This leaves us a high dimensional, non-convex, discrete and continuous space to plan through.

Contact-implicit trajectory optimization (CITO) [1] directly solves nonlinear programming (NLP) problems in the complex hybrid space. Most methods simplify the problems to make them tractable. Simplifications include simple primitive shape representation[15], [16], reducing to 2D, and small number of contact transitions [17], [18], [19]. Moreover, CITO can be slow and intractable without good initialization.

It is worth to explore the space through global search. To capture the discreteness of manipulation systems, previous research searches through predefined manipulation modes and solve for each mode sequence the whole trajectory using an NLP [20]. For dexterous manipulation, manually defining modes could be engineering heavy or even intractable, as suggested by observations in dexterous grasping [21]. Alternatively, expert knowledge in contacts, like contact formations [22], contact states [23] and contact modes [24], [25] are exploited to efficiently generate rigid body motions between two bodies [26], within a robot hand [27], [28], dexterous pregrasps [29], under environment contacts [30], [31], [32], and with local trajectory optimization combined with high-level planning [33], [34]. Contact modes can help guide automatically generation of motion primitives that are differentiable in dynamics [35]. Built on the idea of exploiting contacts, our work pushes forward towards a more general framework in 3D dexterous manipulation. We exploit hierarchies inspired by previous hierarchical frameworks for 2D manipulation [36], [37]. While we share a similar idea of decomposing the search in object motions and robot contacts, we design different pipelines, efficient exploration of contacts, and new representations to make it feasible for complex 3D scenarios.

Reinforcement learning (RL) is efficient in discovering manipulation skills from direct interactions with the environment, like in-hand manipulation [2], dexterous grasping [38], and multi-step object reorientation [39]. RL faces the same challenges from high-dimensional complex spaces, leading to sample efficiency problems. RL also requires significant training for new tasks, while our planners can be directly used for a new object or environment. For future research, we hope to combine our framework with RL to leverage the strength of both.

B. Monte-Carlo Tree Search

MCTS is a heuristic search algorithm that uses random sampling for efficient exploration. The AlphaGo family of algorithms [40], [41] combines MCTS with deep neural networks, achieving a superhuman level of play in board games like Go. MCTS has also shown it effectiveness in robotic applications such as robot task planning [42], task and motion planning [43], and object rearrangement planning [44], [45]. MCTS has also shown potentials to plan in the large combinatoric space for contacts. Previous works include gait planning for legged robots [46] and robot contact sequence planning given user-designed object trajectories[47]. Based on these works, our work takes one step further in planning not only robot contacts, but also object interactions with environment contacts (exploiting extrinsic dexterity[48]).

The incorporation of MCTS also offers several current and future benefits, including efficient search through vast complex space, potentials for continuous improvement through deep learning and self-exploration [41], and parallelizability to accelerate the search [49].

III. Preliminary: MCTS

Level 1 and 2 use the MCTS skeleton in Algorithm 1. A search tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ contains a set of nodes $\mathcal{V}$ and edges $\mathcal{E}$. A node is associated with a visited state s. An edge is associated with a state transition $s \xrightarrow{a} s'$ through action a.

GROW-TREE iteratively expands the tree following four steps in Figure 2: selection, expansion, rollout, and backpropagation. Selection determines search directions by selecting the next node through a score that balances exploration and exploitation. We employ the idea in AlphaGo [40] —

Algorithm 1 MCTS skeleton

```
1: procedure SEARCH
2:     T ← NEW-TREE
3:     n ← ROOT-NODE(T)
4:     while n is not a terminal node do
5:         GROW-TREE(T, n)
6:         n ← BEST-CHILD(n)
7:     end while
8: end procedure
```

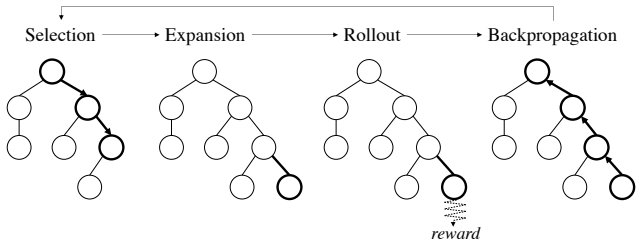

Fig. 2: Four steps of GROW-TREE in MCTS [5]. Selection: start from the root node and select successive explored nodes. Expansion: Create a new node from unexplored actions of the last selected node. Rollout: get the reward of the new node by simulations to the end with random sampling. Backpropagation: Update the tree using rollout results.

use value estimation and action probability to prioritize empirically good directions. Each node maintains a value estimation $v_{est}(s)$, obtained value $v(s)$, and the number of visits $N(s)$. For $s \xrightarrow{a} s'$, we define the action probability $p(s,a)$, the number of visits $N(s,a) = N(s')$, and the state-action value $Q(s,a) = \lambda v(s') + (1-\lambda) v_{est}(s')$, where $\lambda \in [0,1]$ is an adaptive parameter that balance portions of the obtained value and the value estimation. To mitigate the inaccuracy of value estimation, λ increases as the search goes on.

Among the set of feasible actions $\mathcal{A}(s)$, we select the next action $a^* \leftarrow \text{argmin}_{a \in \mathcal{A}(s)} U(s,a)$ with η controlling the degree of exploration:

$$U(s,a) = Q(s,a) + \eta p(s,a) \frac{\sqrt{N(s)}}{1 + N(s,a)} \tag{1}$$

In backpropagation, every node on the evaluated path is updated with the reward r:

$$v(s) = \frac{N(s) v(s) + r}{N(s) + 1} \tag{2}$$

$$N(s) = N(s) + 1 \tag{3}$$

A direct value estimation function is often used to calculate v_{est}. Otherwise if a reward estimation r_{est} is used, we update v_{est} with the same rule as Equation 2.

IV. Hierarchical Planning Framework

A. Task Description

This paper focuses on one rigid body in a non-movable rigid environment or no environment component.

A planner designed under this framework takes in:

1) Object properties: a rigid body $\mathcal{O}$ with known geometry (for example, a mesh model), mass distribution (center of mass and inertia matrix), and friction coefficients with environment μ_{env} and with the manipulator μ_{mnp}.
2) Environment with known geometries.
3) Robot model: The robot manipulates the object through N_{mnp} predefined fingertip contacts. The collision models, forward and inverse kinematics, finger contact models are known. We assume the robot makes non-sliding and non-rolling contacts.
4) Start specification: object start pose $x_{\text{start}} \in SE(3)$.
5) Goal specification: object goal region $X_{\text{goal}} \subset SE(3)$.

It outputs an object configuration trajectory $x(t)$ and a robot control trajectory $u(t)$.

B. Level 1: Planning Environment Contact Modes and Object Trajectories

Level 1 is summarized in Algorithm 2, and visualized in Figure 3 with a block reorientation example. It plans trajectories of environment contact modes and object configurations, which are then passed to Level 2 for planning robot contacts and further evaluated in Level 3. In the Level 1 selection phase, we interleave the search over discrete environment contact modes and continuous object configurations. When a node is selected for expansion and rollout, an RRT method replaces random rollouts to improve exploration efficiency. The output path from the RRT rollout is added to the MCTS, and is passed to Levels 2 and 3 for reward evaluation.

1) Selection — Interleaved Search Over Discrete and Continuous space: The object configuration space SE(3) is continuous, however, the presence of contacts partition it into a complex space. It has many lower-dimensional subspaces with zero probability to be randomly sampled on. For instance, in Figure 3, all object poses for a cuboid pivoting on an edge (mode 0011) lie in a 1D space in the 6D object configuration space. The 1D space has zero probability to be randomly sampled on. We use contact modes for efficient guidance in planning motions in low-dimensional manifolds. A contact mode describes the relative motion of each contact in the system, which is either "maintain" (0) or "separate" (1).

We define Level 1 state as $s1 = (x \in \text{SE}(3), nodetype \in \{mode, pose\})$, where x is an object pose and *nodetype* stores the type of the node.

The interleaved selection process is demonstrated in line 6-25 in Algorithm 2 and in Figure 3. For a *pose* node, the action is to select a contact mode for it. The feasible actions are $\mathcal{A}\big(s1 = (x, pose)\big) = \mathcal{M}(x)$, where $\mathcal{M}(x)$ is the set of kinematically feasible contact modes enumerated for an object pose x using the algorithm in [25]. The result from assigning a contact mode to a *pose* node is a *mode* node. For a *mode* node, the action is to select the next object pose moving from the current object pose following the contact mode. The available actions $\mathcal{A}\big(s1 = (x, mode)\big)$ comprise choosing from its child nodes (explored object poses) or

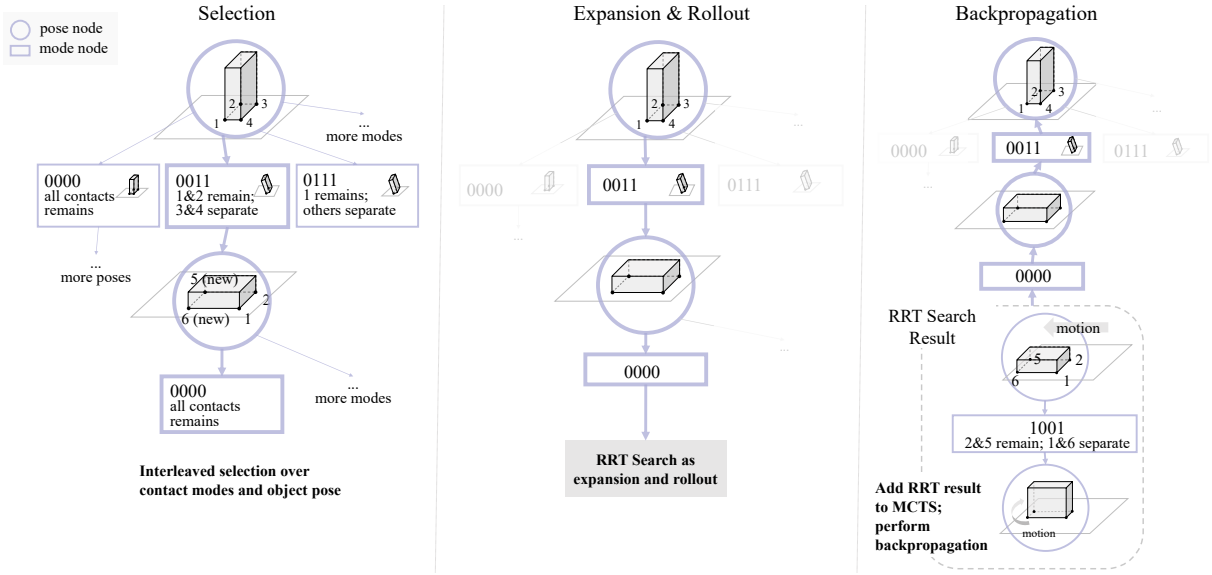

Fig. 3: Level 1 search visualized with a block reorientation example. In the selection phase, the block is selected to follow contact mode 0011, leading to a 90-degree rotation at the edge of contact 1 and 2. Contact mode 0000 (all contacts maintain), is then selected for expansion and rollout. The RRT finds a solution where the block rotates 90 on the edge of contact 2 and 5. New nodes from the RRT solution path are added to the MCTS. The reward is evaluated and backpropagated.

explore-new, which triggers the expansion and rollout phases to explore new object poses.

The selection policy follows Equation 1. Action probabilities $p(s,a)$ reflect preferences of modes or poses. For example, if we prefer to exploit environment constraints to reduce uncertainties as in [12]), we could design probability functions that prioritize modes that maintain contacts.

2) Expansion: The expansion phase equals to *explore-new* being selected for a *mode* node. It is a variant of the progressive widening technique in MCTS for continuous space [50], where we control the expansion rate with the action probability of *explore-new*.

3) RRT as Rollout: In a traditional MCTS, a new node is added by sampling an unexplored action and then evaluated by sampling random rollouts to the end. As our search space has a continuous part and is high-dimensional with sparse solutions from lower-dimensional manifolds, it is unlikely to get any useful results from a random trajectory rollout. We replace the random rollout with an RRT search guided by contact modes (line 17, Algorithm 2), modified based on [31]. According to [51], "an RRT can be intuitively considered as a Monte-Carlo way of biasing search". Our treatment fits the Monte-Carlo philosophy while improved with better guidance towards the goal.

Here is a brief description of the RRT. Details can be found in Appendix III. The *mode* node to expand provides the RRT with the current object pose x_{current} and selected contact mode m_{selected}. The RRT tries to reach x_{goal}. It outputs a trajectory where each point is an object pose associated with a contact mode. In each iteration, we first sample an object pose $x_{extend} \in \text{SE}(3)$ and find its nearest neighbor x_{near}. We then extend x_{near} towards x_{extend}. Each extension is under the guidance of a contact mode. If x_{near} is x_{current}, the contact mode should be m_{selected}. Otherwise, we enumerate all environment contact modes and filter them using feasibility checks. New object poses are generated by forward integration that follows each feasible contact mode as close as possible to x_{extend}. If the RRT finds a solution to x_{goal} within the maximum number of iterations, we add the solution path after the expansion node in the Level 1 search tree and proceed to Level 2 (line 19, Algorithm 2) to obtain a reward. Otherwise, this process backpropagates zero reward and no new node is added. The RRT is reused throughout the entire lifespan of the MCTS.

Compared to [31], we can turn on the option to relax the feasibility check for a contact mode to be "exist any feasible robot contact" while the previous work maintain robot contacts in its states, searching in a more complex space of $\text{SE}(3) \times \mathbb{R}^{N_{mnp}}$. This relaxation could improve the planning speed for some tasks as discussed in Section V.

C. Level 2: Planning Robot Contacts

Level 2 initiates in EVALUATE-REWARD in Level 1 (line 19 and 27, Algorithm 2). Level 2 takes in the object trajectory, and output the best robot contact sequence. The best reward will be passed back to Level 1. Algorithm 3 summarizes the GROW-TREE process in Level 2 MCTS.

1) State and Action Representation: In Level 2 search tree, each node is associated with a robot contact state $s2 = \big(t, \{(i, p_i) | i \in \text{active fingers at } t\}\big)$. This represents that the robot makes contacts by specified active fingers at contact locations $\{p_i \in \mathbb{R}^3\}$ on the object surface at timestep t. For example, grasping an object at timestep 0 with the first and third finger at locations $(1,0,0)$ and $(-1,0,0)$ can be written as $\big(0, (1, (1,0,0)), (3, (-1,0,0))\big)$.

Algorithm 2 Level 1: Search Object Motion

```
1: procedure GROW-TREE-LEVEL-1(𝒯, startnode)
2:     while resources left do
3:         n ← startnode
4:         ▷ Interleaved selection over pose and mode
5:         while n is not terminal do
6:             if NODETYPE(n) is pose then
7:                 ▷ [Selection] next contact mode
8:                 𝒜(n) ← feasible contact modes of n
9:                 a ← SELECT(𝒜(n))
10:                n ← MODE-NODE(a)
11:            end if
12:            if NODETYPE(n) is mode then
13:                ▷ [Selection] next pose
14:                𝒜(n) ← CHILDREN-OF(n) ∪ explore-new
15:                a ← SELECT(𝒜(n))
16:                if a is explore-new then                ▷ [Expansion]
17:                    path ← RRT-EXPLORE(n)               ▷ [Rollout]
18:                    ATTACH(𝒯, path)
19:                    r ← EVALUATE-REWARD(path)           ▷ To Level 2
20:                    BACK-PROPAGATE(r, 𝒯)                ▷ [Backpropagation]
21:                    break loop
22:                else
23:                    n ← TO-NEXT-NODE(a)
24:                end if
25:            end if
26:        end while
27:        r ← EVALUATE-REWARD(n)                          ▷ To Level 2
28:        BACK-PROPAGATE(r, 𝒯)                            ▷ [Backpropagation]
29:    end while
30: end procedure
```

Algorithm 3 Level 2: Search Robot Contacts

```
1: procedure GROW-TREE-LEVEL-2(𝒯, startnode)
2:     while resources left do
3:         n ← startnode
4:         while n is not terminal do
5:             𝒜_sp(n) ← SAMPLE-FEASIBLE-ACTIONS(n)
6:             a ← SELECT(𝒜_sp(n))
7:             n ← NEXT-NODE(n, a)
8:         end while
9:         r ← EVALUATE-REWARD(n)                          ▷ To Level 3
10:        BACK-PROPAGATE(r, 𝒯)
11:    end while
12: end procedure
```

Each action $a = \left(t_c, \{(j, p_j) | j \in \text{relocating fingers at } t_c\}\right)$ represents robot contact relocations, specified by relocating timestep t_c, relocating fingers, and the contact points they are relocating to $\{p_j\}$. Continuing the last example, if we choose to maintain the grasp until timestep 4, and then move the third finger to $(0, 0, 1)$ and add the fifth finger at $(-1, 0, 0.5)$, the action is written as $\big(4, (3, (0, 0, 1)), (5, (-1, 0, 0.5))\big)$. The resulting new state is $\big(4, (1, (1, 0, 0)), (3, (0, 0, 1)), (5, (-1, 0, 0.5))\big)$.

Compared to the common practice of planning contacts for every timestep [19], [47], we plans for contact relocations. While the complexity of the search space does not change, empirically in most tasks, this modification significantly reduces the depth of the search tree and speeds up the discovery of a solution (experiments in Section V, Figure 5).

2) Sampling and Pruning for Action Selection: If we consider 100 object surface contact points, 4 fingers, and a 10-step trajectory. The action space is $100^4 * 10 = 1,000,000,000$. Thus it is not practical to evaluate and compare all actions. To mitigate this issue, we adopt action sampling techniques that are efficient in non-enumerable action space [52].

In Equation 1, we consider a subset of all available actions $\mathcal{A}_{\text{sp}}(s2) \subset \mathcal{A}(s2)$. $\mathcal{A}_{\text{sp}}(s2)$ includes all previously explored actions and newly sampled actions. The newly sampled actions are generated by first sampling the relocating timestep t_c, and then which robot contact(s) to relocate to what location(s) on the object surface. The relocating timestep t_c is sampled in $(t, t_{max}]$, where t_{max} is the maximum timestep the current set of contacts can proceed to under the feasibility check in Section IV-C.3. After t_c is sampled, we sample a robot contact relocation through rejection sampling. We first find relocatable robot contacts by checking if the fingers left satisfy the relocation conditions. Then we sample a feasible new contact location for each relocating finger on the object surface.

If the selected action is an explored action, we are performing the selection phase. If the selected action is a new action, we are performing the expansion and rollout phase. We mix the selection, expansion, and rollout using the same heuristic function. However, one can also use different rollout policies or use value estimation to completely replace the rollout.

3) Feasibility Check: To prune fruitless search directions, we enforce Level 2 nodes and Level 1 RRT rollout nodes to pass the feasibility check, including:

- Kinematic feasibility: whether there exist inverse kinematics solutions for the robot contact points
- Collision free: whether the robot links are collision-free with the environment and the object
- Relocation feasibility: whether there exists a plan to relocate from previous robot contacts to new contacts
- Force conditions: whether the chosen contact points can fulfill the task dynamics, for example, we use a quasistatic or quasidynamic model for our test tasks with environment interactions, and force balance or force closure conditions for in-hand manipulation.
- Other task-specific requirements may also be added.

D. Level 3: Path Evaluation and Control Optimization

Level 2 passes a full path including object motions and robot contact sequence. Level 3, called in line 9, Algorithm 3, checks the feasibility and computes the robot controls $u(t)$. Level 3 can take different methods as long as it provides the reward and robot controls.

For example, if the task mechanics are quasi-static or force-closure, as the timing does not matter on the optimization side anymore, "timestep" becomes "step". For each step t, we could individually solve to check whether quasi-static or force-closure solutions exist and output the robot positions and optimal contact forces as the control $u(t)$, which can potentially be executed using hybrid force velocity control methods [53]. If full dynamics is required, we could

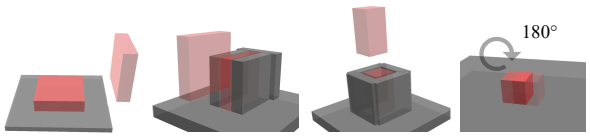

Fig. 4: Scenarios for manipulation with extrinsic dexterity. From left to right: Scenario 1, *pick up a card*: pick up a thin card that cannot be directly grasped using two robot contacts. Scenario 2, *bookshelf*: get a book among other books out of a bookshelf using three robot contacts. Scenario 3, *peg-out-of-hole*: get a peg out of a tight hole using three robot contacts (narrow gaps prevent direct grasps). Scenario 4, *block flipping*: flip the block 180 degrees in the table using two robot contacts.

potentially use the path to be evaluated as a warm-start for trajectory optimization methods [33], [54] to find the control outputs and locally improve the trajectory.

For the resultant control trajectory $u(t)$, we compute the reward r and the estimations v_{est} or r_{est}. There are two rules for defining a reward function: 1) A feasible path should have a positive reward. A non-feasible path should have a zero or negative reward. It is preferred that the reward is in $[0, 1]$. 2) There should be a term that regularizes the length of the path. Otherwise, the search might never end.

V. Examples and Experiments

We implemented two task types: manipulation with extrinsic dexterity and in-hand manipulation. In our code, setting up new scenarios and adjusting search parameters only require modification of one *setup.yaml* file.

A. Implementation

We use Dart [55] as the visualization tool and Bullet [56] for collision detection. We include detailed setup requirements in Appendix I and more implementation details in Appendix II.

1) Robot Model: We implemented two robot types.

Ball fingertips: Each fingertip is a sphere with workspace limits as kinematic feasibility check. We check for collision of the sphere and the environment. We use three vertices of an equilateral triangle on the sphere perpendicular to the contact normal to approximate a patch contact.

Dexterous Direct-drive Hand (DDHand): A DDHand has two fingers. Each fingertip has two degrees of freedom for planar translation and is equipped with a horizontal rod [57], [58]. We provide analytical inverse kinematics model and use line contact model as the fingertip contact model.

2) Task Mechanics: We implemented quasi-static, quasi-dynamic and force closure models. For each timestep, we solve a convex optimization problem to find if contact force solutions exist (details in Appendix II-A.2).

3) Feasibility Checks: include task mechanics check, finger relocation force check (during relocation, it needs to satisfy the task mechanics assuming the object is static), kinematic feasibility check, and collision check.

Scenario	1		2		3		4	
Solution found time (s)	**5.1**	11	**1.2**	7.5	**5.9**	17	**4.6**	10
Success rate	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Nodes in tree	**108**	59	**165**	33	**110**	152	**813**	27
Solution length	**6.2**	9.7	**6.0**	7.4	**11**	17.0	**12**	6.5
Travel distance ratio	**1.1**	1.4	**1.0**	1.2	**1.1**	1.8	**1.8**	1.2
Finger relocations	**1.1**	4.1	**1.0**	3.9	**3.3**	4.7	**2.6**	4.5
Env contact changes	**2.4**	2.9	**2.0**	2.7	**4.6**	4.5	**3.2**	4.7
Grasp centroid distance	**0.2**	0.5	**0.9**	1.1	**0.6**	0.4	**0.3**	0.9

TABLE I: Planning statistics for manipulation with extrinsic dexterity: our method (left, bold), CMGMP (right).

4) Features and Rewards: We use features including travel distance ratio (total object travel distance divided by the start to goal distance), path size, robot contact change ratio (number of finger contact changes divided by the path length), and grasp centroid distance [59]. Given some feature values as data points, we manually label the reward values, favoring smaller object traveling distance, less number of contact changes, and better grasp measures. Given the labeled data, we fit a logistic function as the reward function.

5) Action Probability: In Level 1, in choosing the next contact mode, the action probability prioritizes the same contact mode as before:

$$p\big(s1 = (x, mode), a\big) = \begin{cases} 0.5 & \text{if} \quad a = \text{previous mode} \\ \frac{0.5}{\#\text{modes}-1} & \text{else} \end{cases} \tag{4}$$

In choosing the next configuration, we use a uniform distribution among all the children plus the expansion action.

In Level 2, in choosing a timestep to relocate and the contact points to relocate to, the action probability is calculated using a weight function $w(s2, a)$

$$p(s2, a) = \frac{w(s2, a)}{\sum_{a' \in \mathcal{A}_{\text{sp}}(s2)} w(s2, a')} \tag{5}$$

$w(s2, a)$ encourages previous robot contacts to stay until $t_{\max}$:

$$w(s2, a) = \begin{cases} 0.5 + \frac{0.5}{t_{\max} - t_c + 1} & \text{if} \quad t_c = t_{\max} \\ \frac{0.5}{t_{\max} - t_c + 1} & \text{else} \end{cases} \tag{6}$$

6) Value Estimation: We only use value estimation in Level 1. Each node has $v_{est} = 0.1$ if any Level 2 search can find a valid robot contact sequence for it.

7) Search Parameters: We let $\eta = 0.1$ for both levels. We set λ to 1 if a positive reward is found, otherwise 0.

B. Manipulation with Extrinsic Dexterity

We evaluate four examples in Figure 4. Each scenario is implemented with ball fingertip model without workspace limit and quasi-static mechanics. Additional scenarios are demonstrated in the real robot experiments in Section V-D.

Table I shows the planning statistics from 100 runs for each scenario, using a desktop with the Intel Core i9-10900K 3.70GHz CPU (also for all other statistics in this paper). As our algorithm is anytime, we let the planner stop after 10 seconds and collect the results.

Fig. 5: Planning time (in log scale) with respect to search space size for planning robot contacts for cube sliding. We have *search space size* = *candidate contacts*$^{\textit{trajectory size}}$.

1) Ablation of Hierarchical Structure and MCTS: We compare the results with CMGMP [31], which uses a single RRT in searching for object motions and robot contacts. For all scenarios, our method finds solutions faster. Since the MCTS takes effect after a solution is found, the main contribution in speed is from the hierarchical structure, which decouples the search space of object poses and robot contacts and enable faster solution discovery. Compared to CMGMP, our method also finds solutions with smaller travel distances, less finger relocation, and smaller grasp centroid distance, as guided by the MCTS rewards.

2) Efficient Robot Contact Planning: Level 2 improves robot contact planning by planning contact relocations, while the common practice plans contacts for each timestep on the trajectory [47], [19] (w/o relocation selection). In this ablation study, we consider robot contact planning on a straight line cube sliding trajectory with one allowable robot contact. We run the planning under different numbers of total timesteps and candidate object surface points. As shown in Figure 5, the search space size grows exponentially for both methods. The planning time of the common practice also grows exponentially. As comparison, our modification uses drastically smaller time. Our assumption is that this modification aligns better with the nature of manipulation — contact relocations are sparse compared to the timesteps of the entire trajectory.

C. In-hand Manipulation

A robot can use its fingers to shift and reorient an object to a desired pose within the hand. As the workspace of fingertips is often limited to very small ranges, complex motions are needed. In-hand manipulation demonstrate the effective use of intrinsic dexterity of the robot itself.

1) Different Hand Configurations: We test on three hand setups and object models from the YCB dataset [60]. Here we do not consider the collision of finger links. If needed, robot link models should be provided. For inherently safer motions, we require every motion to have force balance or force closure solutions. Table II shows the statistics for each task with 100 runs of randomized start and goal object poses. Without any training or tuning, our framework achieves a high planning success rate within seconds. Point sampling on the object surface ensures consistent performance for complex object shapes, demonstrated by the ability to plan contacts inside concave objects, such as the mug and the power drill, as shown in the video.

2) Add an Auxiliary Goal: While our framework is designed for object poses as goal specifications, here we demonstrate that it is possible to incorporate auxiliary references, like fingertip locations. We first define d_c, the average robot contact distance to the goal divided by an empirical characteristic length (for example, the object length). We fit a new reward function that prefers small d_c. We then bias the action probability to sample contact locations that are closer to the goal through $w(s2, a)$ in Equation 12:

$$w(s2,a) = \begin{cases} 0.5 + \frac{0.5}{t_{\max} - t_c + 1} p_r(d) & \text{if} \quad t_c = t_{\max} \\ \frac{0.5}{t_{\max} - t_c + 1} p_r(d) & \text{else} \end{cases} \tag{7}$$

We compare planners with and without additional goal fingertip location for 100 reorientation trials with a hammer and a mug using a 5-finger hand. Each trial has a randomized start pose, goal pose, and goal fingertip locations. As Table III shows, the planner with the additional goal specification results in smaller "Final finger distance", but more finger relocations are needed. Other features remain less affected. Due to potential conflicts from the primary goal and trade-offs from other reward terms, there is no guarantee to achieve good alignment with the auxiliary goal.

D. Robot Experiments

We test 8 new scenarios on a dexterous direct drive hand (DDHand) [57] and a configurable array of soft delta robots (delta array) [61]. For both systems, we perform open-loop execution (no object pose estimation or contact feedback). Given a planned fingertip trajectory, we compute the robot joint trajectory using inverse kinematics and execute it with joint position control. The object start position errors are calibrated within 1mm for the DDHand and 2cm for the delta array. Figure 7 and Figure 6 show the keyframes. The full recordings are in the supplementary video.

1) DDHand: We show that the planner enables the DDHand to use intrinsic and extrinsic dexterity. For example, in *occluded grasp*, the fixed green block and the table prevent a direct grasp. The DDHand uses three steps: pivot the object on the corner; use one finger to hold the object; move the other finger to the other side form a grasp. In *upward peg-in-hole*, without a grasp, gravity will cause the peg to fall. But the walls of the hole prevent the fingers from getting in while grasping the object. The DDHand uses one finger to press the peg against the hole — using the wall as an external finger to grasp. The other robot finger then relocate to push the peg from the bottom. The pressing finger also releases to create space for the peg to be pushed in.

2) Delta Array: We test on 4 scenarios: *planar passing* of a cuboid with 2 or 6 separated (no workspace overlap) fingers, *planar reorientation* on a table with 5 or 6 adjacent (have workspace overlap) fingers. Due to the small

Hand Type	3 fingers			4 fingers				5 fingers			
Object	tuna fish can	b lego duplo	cylinder	apple	clamp	mug	power drill	banana	b toy airplane	hammer	c lego duplo
Solution found time(s)	12.0 ± 15.3	4.9 ± 5.5	2.8 ± 4.5	0.3 ± 0.2	3.9 ± 4.9	0.6 ± 0.8	3.9 ± 4.7	0.6 ± 0.7	0.6 ± 0.6	0.6 ± 0.5	0.7 ± 1.1
Success rate	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Nodes in Tree	56 ± 52	33 ± 15	50 ± 25	69 ± 29	69 ± 51	61 ± 29	72 ± 38	68 ± 29	70 ± 33	71 ± 30	63 ± 26
Solution length	10.7 ± 3.3	9.1 ± 2.8	8.2 ± 2.5	8.4 ± 2.1	9.1 ± 3.1	7.9 ± 2.4	9.7 ± 3.0	8.4 ± 2.3	8.4 ± 2.4	8.7 ± 2.3	8.4 ± 2.4
Travel distance ratio	1.5 ± 0.4	1.3 ± 0.3	1.1 ± 0.1	1.0 ± 0.1	1.1 ± 0.2	1.0 ± 0.1	1.1 ± 0.2	1.0 ± 0.1	1.0 ± 0.1	1.0 ± 0.1	1.0 ± 0.1
Finger relocation	3.1 ± 1.8	3.1 ± 1.9	3.0 ± 1.7	4.3 ± 1.7	3.4 ± 2.2	3.8 ± 1.7	3.7 ± 2.1	4.3 ± 1.8	4.2 ± 1.9	4.3 ± 1.8	4.4 ± 1.8

TABLE II: Planning statistics for in-hand manipulation for different finger arrangements (workspaces shown by colored boxes) and objects (images from YCB dataset[60], except for the cylinder).

	With additional goal		Without	
	hammer	mug	hammer	mug
Solution found time(s)	0.89	0.50	0.64	0.35
Success rate	1.0	1.0	1.0	1.0
Nodes in tree	97	82	92	78
Solution length	8.7	8.2	8.5	8.0
travel distance ratio	1.03	1.04	1.03	1.04
Finger relocation	**7.4**	**5.7**	**4.9**	**4.1**
Final fingertip distance	**0.88**	**0.70**	**1.41**	**0.93**

TABLE III: Planning statistics for a 5-finger hand reorienting a hammer and a mug with and without additional goal specification of robot contact locations.

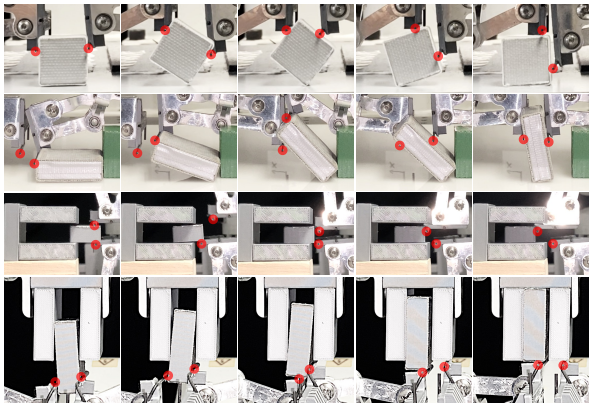

Fig. 6: Keyframes of DDHand experiments. From top to bottom: *cube reorientation*, *occluded grasp*, *sideway peg-in-hole*, *upward peg-in-hole*. The locations of the fingertips are marked with red circles.

workspace of a delta robot (a cylinder with a 2 cm radius and 6cm height), many contact changes are required to accomplish the tasks.

VI. Discussion

This paper proposes a hierarchical framework for planning dexterous robotic manipulation. It facilitates efficient searches across complex spaces, generation of diverse manipulation skills, utilization of expert knowledge, and adaptability for various scenarios. This method has potentials for the automation of wide-ranging manipulation applications, such as functional grasps, caging, forceful manipulation, and mobile and aerial manipulation.

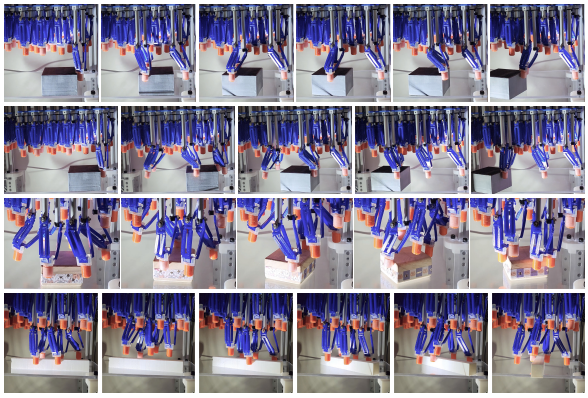

Fig. 7: Keyframes of the delta array experiments. From top to bottom: 2-finger *planar block passing*, 6-finger *planar block passing*, 6-finger *planar reorientation*, 5-finger *planar reorientation* for a long block.

Our framework design allows future extensions to incorporate trajectory optimization and reinforcement learning. By adding dynamic trajectory optimization [62] in Level 3, we could potentially plan dynamic manipulation and smooth object trajectories. Incorporating learning methods is also a future direction. Our method obtains diverse skills with simple hand-coded action policies and rewards. Can past planning experience be leveraged to learn universal contact policies for general manipulation and enhance and be enhanced by reinforcement learning methods?

There are two major limitations when evaluating applicability. First, fast IK methods for robot fingertips are required, which is not direct for tendon-driven or soft robot hands. Second, this framework uses fingertips only, meaning that other part of the robot bodies cannot be used for manipulation. Planning finger rolling is also not supported. We plan to incorporate whole-hand manipulation concepts in future developments to fully leverage robot dexterity.

REFERENCES

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

[2] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, 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.

[3] Y. Hou, Z. Jia, and M. T. Mason, "Fast planning for 3d any-pose-reorienting using pivoting," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1631–1638.

[4] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," *The international journal of robotics research*, vol. 15, no. 6, pp. 533–556, 1996.

[5] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte Carlo tree search: A review of recent modifications and applications," *Artificial Intelligence Review*, pp. 1–66, 2022.

[6] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[7] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[8] Y. Aiyama, M. Inaba, and H. Inoue, "Pivoting: A new method of graspless manipulation of object by robot fingers," in *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*, vol. 1, 1993, pp. 136–143 vol.1.

[9] A. Holladay, R. Paolini, and M. T. Mason, "A general framework for open-loop pivoting," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3675–3681.

[10] X. Cheng, Y. Hou, and M. T. Mason, "Manipulation with suction cups using external contacts," in *Robotics Research: The 19th International Symposium ISRR*. Springer, 2022, pp. 692–708.

[11] Y. Maeda, T. Nakamura, and T. Arai, "Motion planning of robot fingertips for graspless manipulation," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 3. IEEE, 2004, pp. 2951–2956.

[12] C. Eppner, R. Deimel, J. Alvarez-Ruiz, M. Maertens, and O. Brock, "Exploitation of environmental constraints in human and robotic grasping," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 1021–1038, 2015.

[13] C. Eppner and O. Brock, "Planning grasp strategies that exploit environmental constraints," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4947–4952.

[14] J. Z. Woodruff and K. M. Lynch, "Planning and control for dynamic, nonprehensile, and hybrid manipulation tasks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4066–4073.

[15] I. Mordatch, Z. Popović, and E. Todorov, "Contact-invariant optimization for hand manipulation," in *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*. Eurographics Association, 2012, pp. 137–144.

[16] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–8, 2012.

[17] J. Sleiman, J. Carius, R. Grandia, M. Wermelinger, and M. Hutter, "Contact-implicit trajectory optimization for dynamic object manipulation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 6814–6821.

[18] F. H. N. Doshi and A. Rodriguez, "Hybrid differential dynamic programming for planar manipulation primitives," in *ICRA*, 2020.

[19] B. Aceituno-Cabezas and A. Rodriguez, "A global quasi-dynamic model for contact-trajectory optimization in manipulation," 2020.

[20] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning." in *Robotics: Science and Systems*, vol. 2, 2018.

[21] Y. C. Nakamura, D. M. Troniak, A. Rodriguez, M. T. Mason, and N. S. Pollard, "The complexities of grasping in the wild," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 233–240.

[22] J. Xiao and X. Ji, "Automatic generation of high-level contact state space," *The International Journal of Robotics Research*, vol. 20, no. 7, pp. 584–606, 2001.

[23] X. Ji and J. Xiao, "Planning motions compliant to complex contact states," *The International Journal of Robotics Research*, vol. 20, no. 6, pp. 446–465, 2001.

[24] M. T. Mason, *Mechanics of Robotic Manipulation*. Cambridge, MA, USA: MIT Press, 2001.

[25] E. Huang, X. Cheng, and M. T. Mason, "Efficient contact mode enumeration in 3d," in *Workshop on the Algorithmic Foundations of Robotics*, 2020.

[26] P. Tang and J. Xiao, "Automatic generation of high-level contact state space between 3d curved objects," *The International Journal of Robotics Research*, vol. 27, no. 7, pp. 832–854, 2008.

[27] J. C. Trinkle and J. J. Hunter, "A framework for planning dexterous manipulation," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, 1991, pp. 1245–1251 vol.2.

[28] M. Yashima, Y. Shiina, and H. Yamaguchi, "Randomized manipulation planning for a multi-fingered hand by switching contact modes," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 2. IEEE, 2003, pp. 2689–2694.

[29] S. Chen, A. Wu, and C. K. Liu, "Synthesize dexterous nonprehensile pregrasp for ungraspable objects," *arXiv preprint arXiv:2305.04654*, 2023.

[30] X. Cheng, E. Huang, Y. Hou, and M. T. Mason, "Contact mode guided sampling-based planning for quasistatic dexterous manipulation in 2d," *IEEE International Conference on Robotics and Automation*, 2021.

[31] ——, "Contact Mode Guided Motion Planning for Quasidynamic Dexterous Manipulation in 3D," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 2730–2736.

[32] J. Liang, X. Cheng, and O. Kroemer, "Learning Preconditions of Hybrid Force-Velocity Controllers for Contact-Rich Manipulation," *Conference on Robot Learning*, 2022.

[33] T. Pang, H. Suh, L. Yang, and R. Tedrake, "Global Planning for Contact-Rich Manipulation via Local Smoothing of Quasi-dynamic Contact Models," *arXiv preprint arXiv:2206.10787*, 2022.

[34] C. Chen, P. Culbertson, M. Lepert, M. Schwager, and J. Bohg, "Trajectotree: Trajectory optimization meets tree search for planning multi-contact dexterous manipulation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 8262–8268.

[35] E. Huang, X. Cheng, Y. Mao, A. Gupta, and M. T. Mason, "Autogenerated manipulation primitives," *The International Journal of Robotics Research*, p. 02783649231170897, 2023.

[36] G. Lee, T. Lozano-Pérez, and L. P. Kaelbling, "Hierarchical planning for multi-contact non-prehensile manipulation," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 264–271.

[37] B. Aceituno and A. Rodriguez, "A Hierarchical Framework for Long Horizon Planning of Object-Contact Trajectories," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 189–196.

[38] W. Zhou and D. Held, "Learning to grasp the ungraspable with emergent extrinsic dexterity," in *ICRA 2022 Workshop: Reinforcement Learning for Contact-Rich Manipulation*, 2022.

[39] W. Zhou, B. Jiang, F. Yang, C. Paxton, and D. Held, "Learning hybrid actor-critic maps for 6d non-prehensile manipulation," *arXiv preprint arXiv:2305.03942*, 2023.

[40] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[41] 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.

[42] B. Kartal, E. Nunes, J. Godoy, and M. Gini, "Monte carlo tree search for multi-robot task allocation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[43] T. Ren, G. Chalvatzaki, and J. Peters, "Extended tree search for robot task and motion planning," *arXiv preprint arXiv:2103.05456*, 2021.

[44] E. Huang, Z. Jia, and M. T. Mason, "Large-scale multi-object rearrangement," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 211–218.

[45] Y. Labbé, S. Zagoruyko, I. Kalevatykh, I. Laptev, J. Carpentier, M. Aubry, and J. Sivic, "Monte-carlo tree search for efficient visually guided rearrangement planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3715–3722, 2020.

[46] L. Amatucci, J.-H. Kim, J. Hwangbo, and H.-W. Park, "Monte carlo tree search gait planner for non-gaited legged system control," in *2022*

International Conference on Robotics and Automation (ICRA), 2022, pp. 4701–4707.

[47] H. Zhu and L. Righetti, "Efficient Object Manipulation Planning with Monte Carlo Tree Search," *arXiv preprint arXiv:2206.09023*, 2022.

[48] N. C. Dafle, A. Rodriguez, R. Paolini, B. Tang, S. S. Srinivasa, M. Erdmann, M. T. Mason, I. Lundberg, H. Staab, and T. Fuhlbrigge, "Extrinsic dexterity: In-hand manipulation with external forces," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1578–1585.

[49] G. M. B. Chaslot, M. H. Winands, and H. J. van Den Herik, "Parallel monte-carlo tree search," in *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*. Springer, 2008, pp. 60–71.

[50] J. Lee, W. Jeon, G.-H. Kim, and K.-E. Kim, "Monte-carlo tree search in continuous action spaces with value gradients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 4561–4568.

[51] S. M. LaValle, "About RRT," http://lavalle.pl/rrt/about.html.

[52] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver, "Learning and planning in complex action spaces," in *International Conference on Machine Learning*. PMLR, 2021, pp. 4476–4486.

[53] Y. Hou and M. T. Mason, "Robust execution of contact-rich motion plans by hybrid force-velocity control," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 1933–1939.

[54] T. A. Howell, S. L. Cleac'h, K. Tracy, and Z. Manchester, "Calipso: A differentiable solver for trajectory optimization with conic and complementarity constraints," *arXiv preprint arXiv:2205.09255*, 2022.

[55] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "Dart: Dynamic animation and robotics toolkit," *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[56] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[57] A. Gupta, Y. Mao, A. Bhatia, X. Cheng, J. King, Y. Hou, and M. T. Mason, "Extrinsic Dexterous Manipulation with a Direct-drive Hand: A Case Study," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 4660–4667.

[58] A. Bhatia, A. M. Johnson, and M. T. Mason, "Direct drive hands: Force-motion transparency in gripper design," in *Robotics: science and systems*, 2019.

[59] M. A. Roa and R. Suárez, "Grasp quality measures: review and performance," *Autonomous robots*, vol. 38, no. 1, pp. 65–88, 2015.

[60] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Yale-cmu-berkeley dataset for robotic manipulation research," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017.

[61] S. Patil, T. Tao, T. Hellebrekers, O. Kroemer, and F. Z. Temel, "Linear Delta Arrays for Dexterous Distributed Manipulation," *arXiv preprint arXiv:2206.04596*, 2022.

[62] K. Tracy, T. A. Howell, and Z. Manchester, "Differentiable collision detection for a set of convex primitives," *arXiv preprint arXiv:2207.00669*, 2022.

[63] Y.-H. Liu, "Qualitative test and force optimization of 3-d frictional form-closure grasps using linear programming," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 1, pp. 163–173, 1999.

APPENDIX I
SETTING UP NEW SCENARIOS

In this section, we provide an overview of what are required when setting up new scenarios. Please check our code and Appendix II for the actual implementation.

A. Applicability

This framework can be considered for the tasks of manipulating a single rigid body object in a rigid environment. Environment components must be fixed and not movable. It can also be used when there is no environment component (in-hand manipulation). We need known models of the object, the environment, and the robot.

The robot used to manipulate the object needs to have known collision models, and forward and inverse kinematics. The only parts that can be used to manipulate the object are the defined "fingertips" on the robot.

B. Setup a new robot/hand

Setting up a new robot is the most complicated part. Specifically for implementation in our C++ code, a new class need to be written to inherit a pre-defined abstract class ROBOTTEMPLATE. The user need to fill some specific pure virtual functions that covers the following aspects.

1) Contact force models for fingertips: We use the point contact model for kinematics. However, as the force model for point contact might be too limited, we allow the use of other contact force models. The contact force models that currently exist in our implementation includes:

- Point contact
- Patch contact: we first approximate the fingertips using spheres centered at the point contact locations. The radius of the spheres should approximate the radius of the contact patch for each fingertip. We approximate the patch contact using three point contacts at vertices of an equilateral triangle that is perpendicular to the contact normal and on the sphere.
- Line contact: we approximate the line contact model by two point contacts on twp ends of the line segment.

2) Forward and inverse kinematics for fingertips: The users need to provide the forward and inverse kinematics for the fingertips.

Given the FK and IK model, we precompute the workspace for each fingertip. For general robot hands, we first sample joint angles to get the fingertip points in the workspace through forward kinematics, and then compute the convex hulls. While hands might differ, we estimate this process takes about seconds (with C++ implementation).

3) Robot collision model: The users need to provide the collision model of the robot or the fingertips. If it is unlikely for the robot links to collide with the object or the environment, it is be okay to only provide the collision shape for the fingertips, which will make the computation much faster. Otherwise, the user could simply provide a robot URDF model.

4) Contact relocation planner (optional): A contact relocation planner is required for checking whether a collision-free path exists for a finger to relocate to another contact location.

5) Contact sampling on the object surface (optional): It is best that each fingertip are relatively independent on the kinematic side. If not, our random sampling of robot contacts on the object surface might have a very high rejection rate ($> 90\%$). In this case, we need the user to provide a method for the specific robot in order to more efficiently sample robot contacts on the object surface.

6) Trajectory optimizer (optional): For the robots that are under-actuated (like wheeled robots), the users need to provide Level 3 a trajectory optimizer that finds feasible object states, robot states, and robot controls given Level 2 outputs as warm-start trajectories.

C. Setup a new task type

After setting up the new robot, we need to enable the robot to do a certain type of tasks. Two major things to consider are task mechanics and task parameters for planning.

1) Task Mechanics: Task mechanics include the specific requirements and dynamical model required by the task. Do we have to fully exploit the dynamic property of the system? If yes, we need to have a good trajectory optimization algorithm for the manipulation system in Level 3 to ensure the solutions are feasible. If the task dynamics do not involve the integration of velocity and the robot is fully actuated, it is not necessary to provide a trajectory optimization method in Level 3. In these cases, the user only needs to write a function `task-dynamics(object pose, object velocity, contact info, ...)` that solves a one-step optimization problem. Examples include quasi-static, quasi-dynamic, closure methods, planar pushing, etc.

2) Design choices: A new task type requires several design choices to be made and some search parameters to be tuned. Once the choices are made, changing environments and objects in the same task type should not require more tuning. According to our experience, making designs and tuning parameters are relatively low-effort. We have found that the planner is not sensitive to specific numerical values for parameters.

The design choices include task features, action probability design for Level 1 and 2, reward design, and value estimation design for Level 1 and 2.

Task features are used in the action probability and reward. Basic features are path length in MCTS, number of robot contact relocations, and object travel distance. Task-dependent features like grasp measures or environment contact changes can be added to encourage specific behaviors like better grasps and less environment contact switches. For generality, it is important to normalize the features by ensuring similar values for desired behaviors across different environments and objects.

There are three action probability functions we need to define: (1) select a contact mode in Level 1, (2) select a child (configuration node) for a mode node in Level 1,

and (3) select (time to relocate, contacts to relocate to) in Level 2. For (1), we often encourage the use of the same contact mode as the previous one. For (2), we currently mostly use a uniform distribution. For (3), we would like to encourage relocating when the contacts are not feasible the next timestep. However, the definition of these probabilities is entirely up to the user.

To design the reward function, we use a simple approach that requires no tuning. Given some feature values as data points, we first manually label their reward values between 0 to 1 through human intuition. Next, we fit a logistic function to these data points as the reward function.

Manually value estimation is very flexible. The value estimation in our method is often used to encourage the search to visit a node that has been visited but has not found any positive reward. For example, on the way to the goal pose, if an object pose is reachable through a sequence of contacts (check by Level 2), we can assign 0.1 as its value estimation. Our design principle is to give a small number to any node that is more likely to find a solution than others.

3) Parameters: The search parameters include MCTS exploration rates η_1, η_2 in Level 1 and 2. Adaptive parameter for value estimation λ. In all of our experiments, we let $\eta_1 = 0.1, \eta_2 = 0.1$. We let $\lambda = 0$ if a positive reward has not been found and otherwise $\lambda = 1$. While tuning the parameters may slightly improve the performance for specific tasks, we suspect that most of the time this is not a must. However, λ might need some tuning if one day we have better value estimations, like using learned functions.

D. Setup a new environment

If using our preset robots and tasks, the users can easily setup new environments and objects in one file called *setup.yaml*.

When setting up a model for a new environment, it is usually adequate to use primitive geometries such as cuboids, cylinders, and spheres in the simulation environment. The users need to specify the shape parameters and the locations of the primitive shapes.

E. Setup a new object

For a new object, the users need to provide the object mesh or specify the primitive shape. Surface points will be automatically uniformly sampled on the mesh. Each point (p,n) is represented by its location ($p \in R^3$) and its contact normal ($n \in S^3$) in the object frame. It is usually sufficient to sample about 100 points. The computation is usually in milliseconds.

The user also need to provide the object mass, object inertia, and friction coefficients for robot-object and environment-object contacts.

For each new object and environment, the RRT parameters might need some changes, including the range of object positions, goal biased sampling probability, unit extend length, and the weight for rotation for the distance calculation. The RRT parameters does not require careful tuning, as long as they roughly reflect the task requirement. For example, the weight for rotation is good to be set to 1 if the object bounding box range is between 0.1 - 10 and rotation and translation are roughly of equal importance. If the object orientation is not important at all, the weight is good to be set to 0.01 to 0.1. The unit extend length should be larger if the object start and goal are very far from each other, otherwise the planner will be slow. And it should be smaller if the user expect many different maneuvers required for the task.

Extra note: to avoid numerical issues, we usually scale the whole system such that the average length of the object bounding box is in the range of 1 - 10.

Appendix II
Experiment Details

This section includes the details of the experiments in this paper. The first two are pure planning experiments. The latter two are robot experiments.

A. Manipulation with Environment Interactions

1) Robot model: We consider the robots as free-flying balls, meaning that we do not check for kinematic feasibility but do check for collision of the balls and the environment. For the contact force model, we use the patch contact model, described in Appendix I-B.

2) Task mechanics: We use quasi-static or quasi-dynamic models. For each timestep, we solve a convex programming problem to find if there exists a solution for contact force λ_c to satisfy the force conditions. The problem is formulated as follows:

$$\begin{aligned} &\min_{\lambda} \|\epsilon \lambda^T \lambda\| \\ \text{s.t.}\quad &\text{quasistatic or quasidynamic condition} \end{aligned} \tag{8}$$

where $\epsilon \lambda^T \lambda$ is a regularization term on the contact forces.

The quasi-static condition requires the object to be under static force balance for a selected contact mode

$$\left[G_1 h_1, G_2 h_2, \ldots\right] \cdot \left[\lambda_1, \lambda_2, \ldots\right]^T + F_{\text{external}} = 0 \tag{9}$$

where $\left[\lambda_1, \lambda_2, \ldots\right]^T$ are the magnitudes of forces along active contact force directions $\left[h_1, h_2, \ldots\right]^T$ determined by contact modes. $\left[G_1, G_2, \ldots\right]^T$ are the contact grasp maps. F_{external} includes other forces on the object, such as gravity and other applied forces.

Quasidynamic assumption relaxes the requirement for objects to be in force balance, allowing short periods of dynamic motions. We assume accelerations do not integrate into significant velocities. In numerical integration, the object velocity from the previous timestep is 0. The equations of motions become:

$$M_o \dot{v^o} = \left[G_1 h_1, G_2 h_2, \ldots\right] \cdot \left[\lambda_1, \lambda_2, \ldots\right]^T + F_{\text{external}} \tag{10}$$

In discrete time, the object acceleration $\dot{v^o}$ can be written as $\frac{v^o}{h}$, where h is the step size. The object velocity v^o is computed by solving the constrained velocity from the current pose to the goal pose under a contact mode.

3) Feasibility Checks:

- Task mechanics check: is passed if there exist a solution for Equation 8.
- Finger relocation check: during relocation, the non-relocating robot contacts and environment contacts must also satisfy the task mechanics, assuming the object has zero velocity.
- Collision check: the spheres must not collide with the environment.

4) Features: We manually designed the features, as shown in Table IV.

Feature	Description
Path size	node depth in the Level 1 tree
Object travel distance ratio	$\frac{\text{total travel distance}}{dist(x_{\text{start}}, x_{\text{goal}})}$
Robot contact change ratio	$\frac{\text{number of finger contact changes}}{\text{number of fingers}}$
Number of environment contact changes	-
Grasp centroid distance	$dist(c_{contact}, c_{geo})$

TABLE IV: Features for Manipulation with Environment Interactions. $c_{contact}$: the centroid of all contact points; c_{geo}: the geometric center of the object.

5) Action Probability: In Level 1, in choosing the next contact mode, we design the action probability to prioritize choosing the contact mode the same as before:

$$p\big(s1 = (x, mode), a\big) = \begin{cases} 0.5 & \text{if} \quad a = \text{previous mode} \\ \frac{0.5}{\#\text{modes}-1} & \text{else} \end{cases} \tag{11}$$

In Level 1, in choosing the next configuration, we let $p\big(s1 = (x, config), a\big)$ be a uniform distribution for all the children and *explore-new*

In Level 2, in choosing a timestep to relocate and the contact points to relocate, the action probability is calculated using a weight function $w(s2, a)$ designed for each action in $\mathcal{A}_{\text{sp}}(s2)$:

$$p(s2, a) = \frac{w(s2, a)}{\sum_{a' \in \mathcal{A}_{\text{sp}}(s2)} w(s2, a')} \tag{12}$$

The manually designed weight function $w(s2, a)$ prefers to let the previous robot contacts stay as long as possible:

$$w(s2, a) = \begin{cases} 0.5 + \frac{0.5}{t_{\max} - t_c + 1} & \text{if} \quad t_c = t_{\max} \\ \frac{0.5}{t_{\max} - t_c + 1} & \text{else} \end{cases} \tag{13}$$

6) Reward Design: We use all the features in Table IV and follow the logistic function fitting procedure as described in Appendix I-C.2.

7) Value Estimation: We only use value estimation for Level 1 nodes. Each node has $v_{est} = 0.1$ if any subsequent Level 2 search is able to proceed past that node. For all Level 2 nodes, the value estimation is simply zero.

8) Search Parameters: In both Level 1 and Level 2, we let the exploration rate $\eta_1, \eta_2 = 0.1$. Since we only have value estimation for Level 1, there is only one adaptive parameter λ for Level 1 only. When no reward > 0 has been found, $\lambda = 0$. After any positive reward is observed, $\lambda = 1$.

B. In-hand Manipulation

1) Robot model: The setup is the same as Appendix II-A.1. The only difference is that we now have a workspace limit for each finger.

2) Task mechanics: We use quasi-static models (as described in Appendix II-A.2) or force closure [63].

3) Feasibility Check: includes workspace limit check for fingertips, task mechanics check, and finger relocation check.

Features, action probability, reward, value estimation, and search parameters are the same as the Manipulation with Environment Interactions task.

C. Robot Experiment: Dexterous DDHand

1) Dexterous DDHand Overview: Dexterous DDHand is a direct-drive hand with 4 Dofs. It has two fingers and each finger has 2 Dofs for planar translation motions. Each fingertip is a horizontal rod. As a result, we use two endpoints of the rod to approximate the line contact. We provide the planner with the forward and inverse kinematics of the hand. We also provide a contact relocation planner, which follows the object surface (5mm above the object surface) and goes to the new contact location.

2) Feasibility Checks: include inverse kinematics check, collision check, finger relocation force check, finger relocation path check (are there collisions on the relocation path), and task mechanics check.

Task mechanics, features, action probability, reward, value estimation, and search parameters are the same as the Manipulation with Extrinsic Dexterity task.

3) Execution: Given a planned fingertip trajectory, we compute the robot joint trajectory using inverse kinematics and execute it with robot joint position control. In order to ensure some contact force, we shift the end-effector trajectory in the environment contact normal direction for

$$\Delta\text{position} = \frac{\text{Desired contact force}}{\text{Stiffness}} \tag{14}$$

where the stiffness can be tuned due to the direct-drive property.

The execution was conducted in an open-loop manner, meaning that there was no object pose estimation or force control involved. The system was calibrated to ensure that the initial object pose errors are kept within a tolerance of 1 mm. We chose not to provide a formal success rate in our report since this number lacks significance due to its dependency on the accuracy of our manual calibration process. However, as a point of reference, with an initial pose precision of 1 mm, we estimate a success rate of approximately 4 out of 5 attempts.

D. Robot Experiment: Delta Array

1) Delta Array System Overview: The array of soft delta robots is a research platform for the development of multi-robot cooperative dexterous manipulation skills. The system is comprised of 64 soft linear delta robots arranged in an 8x8 hexagonal tessellating grid. Each 3D printed soft delta linkage is actuated using 3 linear actuators to give 3 degrees of translational freedom with a workspace of 3.5cm radius

in the X, and Y axes and 10cm in Z-axis. The links are compliant with high elasticity and low hysteresis, with a soft 3D printed fingertip-like end-effector attached to it. We simplify the workspace of each delta robot to be a cylinder with a 2.5cm radius and 6cm height.

We provide the forward and inverse kinematic models to the planner. While running the planner, the IK check is simplified to a workspace limit check (if the contact point is in the cylinder workspace). We only perform collision checks for the fingertips, not the links. While doing the actual execution of the plans, we use inverse kinematics to calculate the robot joint trajectory from the contact point trajectory. In order to ensure some contact force, we shift the end-effector trajectory in the same way as Equation 14, where the stiffness is manually calibrated.

We relocate contacts by letting the delta robot to leave the contact in the contact normal direction, go around the edge of the workspace, and come to the new contact in its normal direction. The entire plan is executed in open-loop. Although delta robots may not offer a high level of accuracy and repeatability, their passive compliance allows for minor deviations from the planned trajectory to be accommodated.

2) Feasibility Check: include workspace limit check, collision check, task mechanics check.

Task mechanics, features, action probability, reward, value estimation, and search parameters are the same as the Manipulation with Extrinsic Dexterity task.

APPENDIX III
RRT FOR ROLLLOUT

The RRT process is summarized in Algorithm 4. The inputs are the current object pose x_{current}, selected contact mode m_{selected}, and the object goal pose x_{goal}. If it can find a solution, it outputs a trajectory from x_{current} to x_{goal}. Every point on the trajectory is (x, m), where $x \in \text{SE}(3)$ is an object pose, m is an environment contact mode.

At each iteration, SAMPLE-RANDOM-OBJECT-POSE sample a new object pose $x_{\text{extend}} \in \text{SE}(3)$. We find the nearest neighbor x_{near} of x_{extend}, and attempt to extend it towards x_{extend} (line 5 - 15, Algorithm 4). Each extension is performed under the guidance of a contact mode. If x_{near} happens to be x_{current}, we let the contact mode be m_{selected} chosen by Level 1 MCTS. Otherwise, the function SELECT-CONTACT-MODE will select the contact mode(s) to perform the extension under. The procedure EXTEND-WITH-CONTACT-MODE extends x_{near} towards x_{extend} under the guidance of a selected contact mode m through projected forward integration.

Next, we explain all the functions in detail.

SAMPLE-RANDOM-OBJECT-POSE sample a new object pose $x_{\text{extend}} \in \text{SE}(3)$. The probability of x_{extend} being the goal pose is p_{sample}, while the probability of it being a random object pose in SE(3) is $1 - p_{\text{sample}}$. We can specify the range limit for the random sample of the object pose.

NEAREST-NEIGHBOR finds the closest object pose to x_{extend} in the tree. The distance between two object poses is computed as $w_t * d_t + w_r * d_r$. w_t and w_r are the weights for translation and rotation. d_t is the Euclidean distance between their locations, and d_r is the angle difference between two rotations. A simple way to compute d_r is to first compute the rotation between two poses $R_{\text{diff}} = R_1 R_2^T$, and convert R_{diff} to axis-angle representation and let d_r be equal to the angle. In general, the users need to adjust the weights according to how important object orientation or position is important in the task. Not much tuning is needed. In our experiment, we scale the object sizes such that the average length of their bounding boxes is about 1 to 10. In this case, one can set the weights using this rule: normal (1), not very important (0.5), not important at all (0.1).

SELECT-CONTACT-MODE first enumerates all contacting-separating contact modes, then filters out infeasible mode through EXTEND-FEASIBILITY-CHECK, and finally returns the set of all feasible modes.

Algorithm 4 RRT for Expansion and Rollout

1: **procedure** RRT-EXPLORE(x_{current}, m_{selected}, x_{goal})
2: **while** resources left and the goal is not reached **do**
3: $x_{rand} \leftarrow$ SAMPLE-RANDOM-OBJECT-POSE($x_{goal}, p_{\text{sample}}$)
4: $x_{\text{near}} \leftarrow$ NEAREST-NEIGHBOR(x_{rand})
5: **if** $x_{\text{near}} = x_{\text{current}}$ **then**
6: $\mathcal{M} \leftarrow \{m_{\text{selected}}\}$
7: **else**
8: $\mathcal{M} \leftarrow$ SELECT-CONTACT-MODES($x_{\text{near}}, x_{\text{rand}}$)
9: **end if**
10: **for** $m \in \mathcal{M}$ **do**
11: $x_{\text{new}} \leftarrow$ EXTEND-WITH-CONTACT-MODE($x_{\text{near}}, x_{\text{rand}}, m$)
12: **if** $x_{\text{new}} \neq$ null **then**
13: ADD-TO-RRT-TREE(x_{new}, $\mathcal{T}_{\text{rrt}}$)
14: **end if**
15: **end for**
16: **end while**
17: solution-path $\leftarrow$ BACKTRACK($x_{\text{goal}}, \mathcal{T}_{\text{rrt}}$)
18: **return** solution-path
19: **end procedure**
20: **procedure** SELECT-CONTACT-MODE($x_{\text{near}}, x_{\text{rand}}$)
21: $p_{env} \leftarrow$ ENVIRONMENT-CONTACT-POINT-DETECTION(x_{near})
22: $\mathcal{M}_{env} \leftarrow$ ENUMERATE-CONTACT-MODES(p_{env})
23: **for** all $m \in \mathcal{M}_{env}$ **do**
24: **if** EXTEND-FEASIBILITY-CHECK(m, $x_{\text{near}}, x_{\text{rand}}$) **then**
25: $\mathcal{M} \leftarrow \{M, m\}$
26: **end if**
27: **end for**
28: **return** $\mathcal{M}$
29: **end procedure**
30: **procedure** EXTEND-WITH-CONTACT-MODE($x_{\text{near}}, x_{\text{rand}}, m$)
31: $x_{\text{now}} = x_{\text{near}}$
32: **while** true **do**
33: **if** not EXTEND-FEASIBILITY-CHECK(m, $x_{\text{now}}, x_{\text{rand}}$) **then**
34: break
35: **end if**
36: $v \leftarrow$ VELOCITY-UNDER-MODE(m, $x_{\text{now}}, x_{\text{rand}}$)
37: **if** v close to zero **then**
38: break
39: **end if**
40: ▷ Projected forward integration
41: $x_{\text{now}} \leftarrow$ INTEGRATE(x_{now}, v)
42: $x_{\text{now}} \leftarrow$ PROJECT-TO-CONTACTS-MAINTAINED(x_{now}, m)
43: **if** encounter new contacts **then**
44: break
45: **end if**
46: **end while**
47: **return** x_{now}
48: **end procedure**

EXTEND-FEASIBILITY-CHECK has two options in implementation. The first option involves storing the robot contacts during the search process. If the current robot contacts pass the feasibility check in Section IV-C.3, the check is deemed successful. However, if they fail, the check can still be considered successful if a sampled set of feasible robot contacts can be generated, ensuring a feasible transition from the current contacts. The current contacts are then updated accordingly. The second option finds if there exists a set of robot contacts that satisfy the feasibility check. Unlike the first option, this method does not retain information about robot contacts. Instead, it is consider successful if it can sample any set of robot contacts, as long as they pass the feasibility check. The second option is more relaxed as it does not take into account previous robot contacts and transitions.

EXTEND-WITH-CONTACT-MODE extends x_{near} towards x_{extend} as much as possible under constraints posted by contact mode m. VELOCITY-UNDER-MODE solves for the object velocity that get x_{near} as close as possible to x_{extend} with respect to the velocity constraints introduced by m. We then integrate the object pose for a small step in the direction of constrained object velocity, and project the new object pose back to the contacts that needed to be maintained.

In PROJECT-TO-CONTACTS-MAINTAINED, the contact mode m needs to be maintained. We first perform contact detection on the object. We then project the object pose back to where the maintaining contacts in m have zero signed distances.